

# Developing Plugin Programs

---

## In this chapter . . .

["Understanding plugin programs" on page 443](#)

["Using the API documentation" on page 444](#)

["Developing plugin programs" on page 458](#)

---

Users who know how to program in Java can write a *plugin program* that adds support for a new file format, creates a new view, or applies a new algorithm to an image. This chapter does not explain how to write a Java program; rather it presents information to help users who are writing plugin programs to customize MIPAV. This chapter explains how to do the following:

- Gain access to and use the online MIPAV application programming interface (API) documentation
- Determine which version of Java to use
- Select one of the three plugin types
- Include mandatory lines of code in plugin programs so that they interface correctly with MIPAV
- Install plugin programs

## Understanding plugin programs

Plugin programs, also known simply as *plugins*, are utilities or sets of instructions that add functionality to a program without changing the program. In MIPAV you use Java to write and compile plugin programs to perform specific functions, such as automatically removing all odd-numbered images from the image dataset or adding support for a new file format. There are three types of plugin programs that you may write for MIPAV:

- **Algorithm**—An algorithm type of plugin performs a function on an image. An example is a plugin that applies a radial blur algorithm to an image.



**Note:** You can create plugin algorithms through Java.

- **File**—A file type of plugin allows MIPAV to support a new file format. An example is a plugin that allows MIPAV to view Kodak Photo CD files (.pcd).
- **View**—A view type of plugin introduces a new view, or the way in which the image is displayed. Examples include the lightbox, triplanar, and animate views.



**Note:** Because MIPAV already supports a large number of file formats and views and its development team makes it a practice to extend its capabilities in these areas, it is generally unnecessary to add file or view types of plugins. Most plugin programs, therefore, are algorithms.

After developing a plugin program, you can then install the plugin program into the MIPAV application and access it from the Plugins menu (Figure 298) in the MIPAV window. The MIPAV window labeled “(A)” in Figure 298 shows the Plugins menu as it appears before any plugin programs are installed. The picture labeled “(B)” in Figure 298 shows the Plugins menu as it appears after two plugin programs—in this case, the Fantasm plugin program and the Talairach Transform plugin program—are installed. Because the Fantasm and Talairach Transform plugin programs are algorithms, they appear on the Plugins > Algorithm menu.



**Note:** If a plugin program is a file type of plugin, it would appear under a Plugins > File menu. If it is a view type, it would appear under a Plugins > View menu.

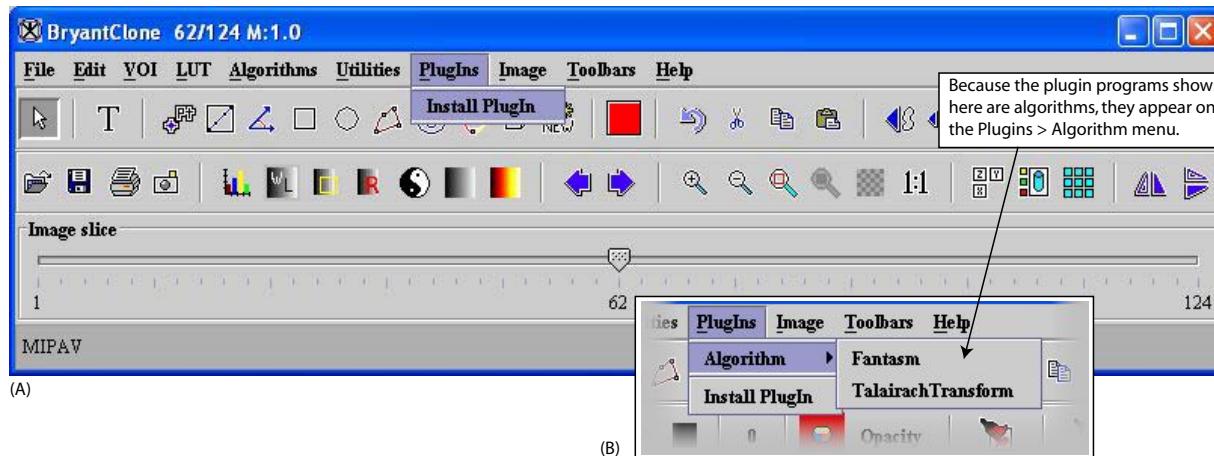


Figure 298. Plugins menu in the MIPAV window: (A) Before a plugin is installed and (B) after two algorithm plugins are installed

## Using the API documentation

Documentation for the application programming interface (API) is located on the MIPAV web site. You can use the documentation directly on the web site. However, if your internet access is limited or slow, you can download, install, and use either a zipped version of the documentation on a Windows workstation or a tar version on a Unix workstation.

### To access the API documentation via the internet

- 1 Go to the MIPAV web site:

<http://mipav.cit.nih.gov/>

- 2 Click Documentation in the links on the left side of the page. The Documentation page appears.
- 3 Under MIPAV Documentation for the developer, notice the following links:

- Application Programming Interface ([JavaDoc](#))
  - [XML Format Guide](#)
- 4** Click [JavaDoc](#). The MIPAV API Documentation page (Figure 299) appears.
- 5** Click [Application Programming Interface JavaDoc](#). The API documentation page (Figure 299) appears.



Figure 299. MIPAV API Documentation page on the MIPAV web site

### To download and install the API documentation on a Windows workstation

- 1** On the second line under Documentation for download, select [available to download](#) a zip-formatted version.
- 2** Save the file to a directory of your choice.

- 
- 3** Go to the directory, double-click api.zip, and extract the files. Extraction creates a directory named “api” under the directory you chose to place the files.
  - 4** Open the api directory, and double-click index.html. The API documentation appears in your browser.

### To download and install the API documentation on a Unix workstation

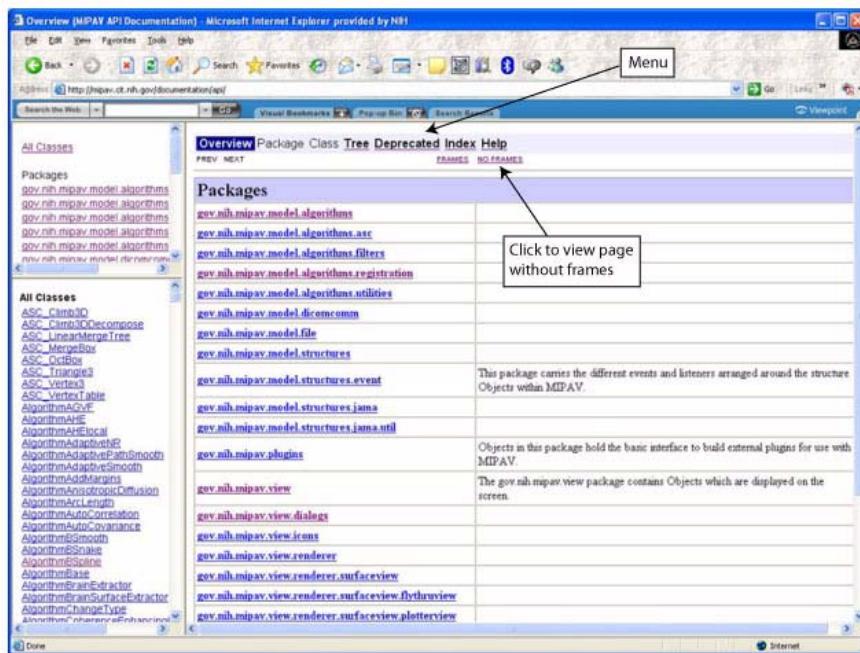
On the first line under Documentation for download, select [available to download](#) a tar.gz-formatted version.

- 1** On the second line under Documentation for download, select [available to download](#) a zip-formatted version.
- 2** Save the file to a directory of your choice.
- 3** Go to the directory, double-click api.zip, and extract the files. Extraction creates a directory named “api” under the directory you chose to place the files.
- 4** Open the api directory, and double-click index.html. The API documentation appears in your browser.

As it appears initially, the API documentation page (Figure 300 on page 447) shows the following three frames:

- **Top left frame**—Shows all of the Java packages for the MIPAV application. When you select the All Classes link at the top of this frame, all of the classes in MIPAV appear in alphabetical order in the bottom left frame. If you select a particular package, the bottom left frame displays only the classes that pertain to the selected package.
- **Bottom left frame**—Lists either all of the classes in the MIPAV application or all of the classes in a selected package.
- **Right frame**—Displays information based on the command that you select in the menu at the top of the frame:
  - **Overview**—Lists all of the packages in the MIPAV application
  - **Package**—Lists and summarizes all of the classes and interfaces in the package
  - **Class or Interface**—Lists descriptions, summary tables, and detailed member descriptions

- **Tree**—Displays a hierarchy of the class or package
  - **Deprecated**—Lists deprecated APIs
  - **Index**—Provides an alphabetical list of all classes, interfaces, constructors, methods, and fields
  - **Help**—Provides help for the API documentation



**Figure 300.** The Overview page in the API documentation

Several links appear beneath the menu.

- **Prev and Next**—These links take you to the next or previous class, interface, package, or related page.
  - **Frames and No Frames links**—These links show and hide the HTML frames. All pages are available with or without frames.

---

## OVERVIEW PAGE

The Overview page is the page that initially appears when you gain access to the API documentation. This page displays a list of all of the packages in MIPAV.

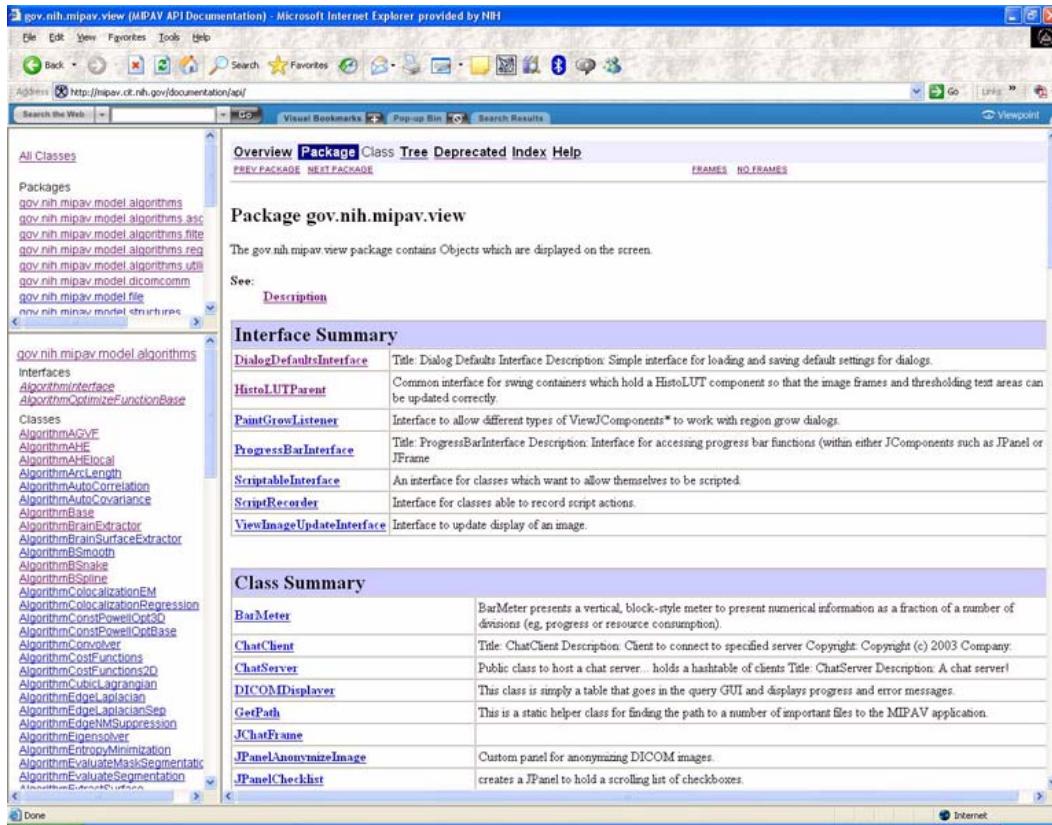
The Overview command on the menu becomes available after you move to another page. For example, if you select a package on the Overview page, the Package page appears. On the Package page, the Overview command becomes available. To return to the Overview page from the Package page, click Overview.

When you click Overview, the Overview page (Figure 300 on page 447) appears and displays a list of all of the packages in MIPAV.

---

## PACKAGE PAGE

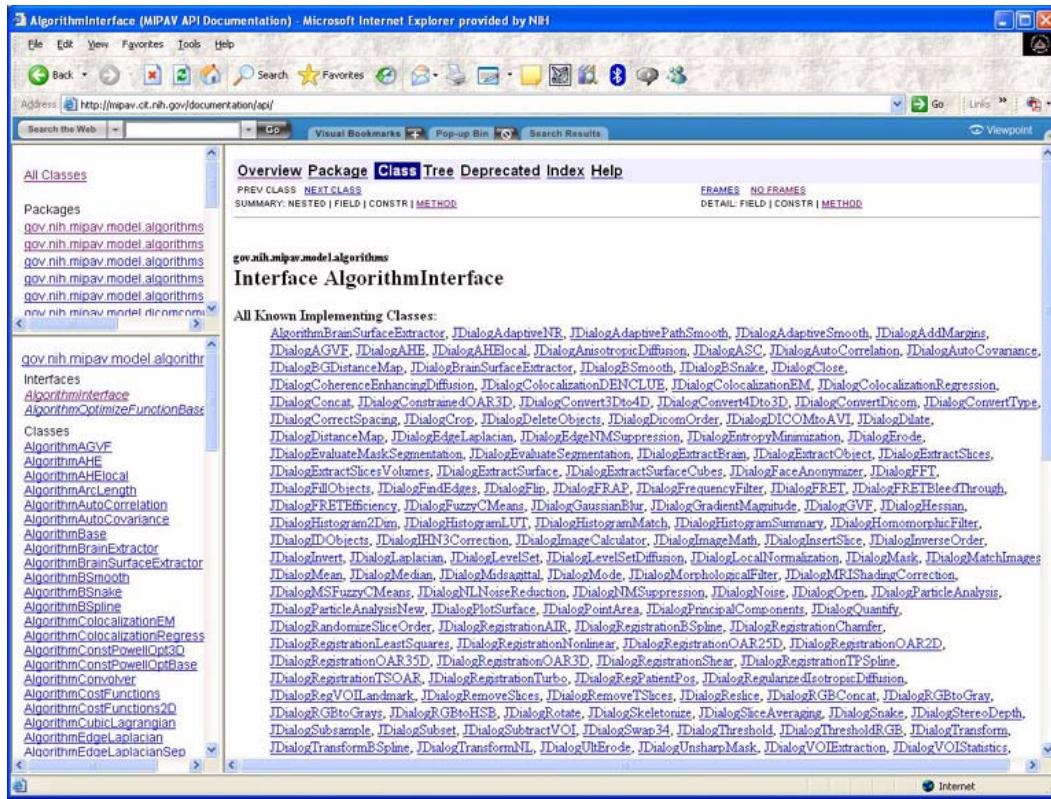
When you select one of the packages listed on the Overview page, the Package page (Figure 301 on page 449) appears. This page provides a summary of each interface (if any), class, and exception (if any) in the package. When you click an interface or class, the Interface page or the Class page appears. Clicking an exception displays the Exception page.



**Figure 301. The Package page in the API documentation**

## INTERFACE OR CLASS PAGES

When you select an interface or class on the Package page, either the Interface page (Figure 302 on page 450) or the Class page (Figure 303 on page 451) appears. Each interface, nested interface, class, and nested class has its own separate page.

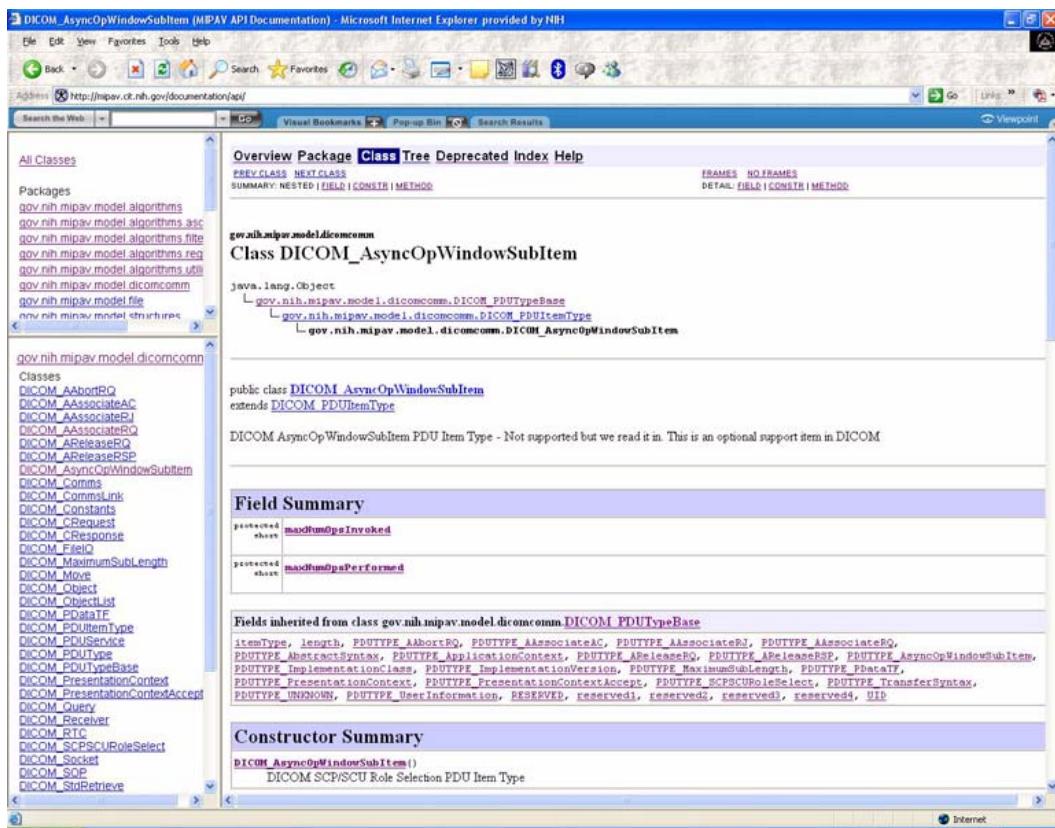


**Figure 302. Interface page in the API documentation**

Each of these pages has three sections consisting of an interface or class description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct known subclasses
- All known subinterfaces or subclasses
- All known implementing classes
- Interface or class declaration
- Interface or class description
- Nested class summary
- Field summary
- Constructor summary

- Method summary
- Field detail
- Constructor detail
- Method detail



**Figure 303. Class page in the API documentation**

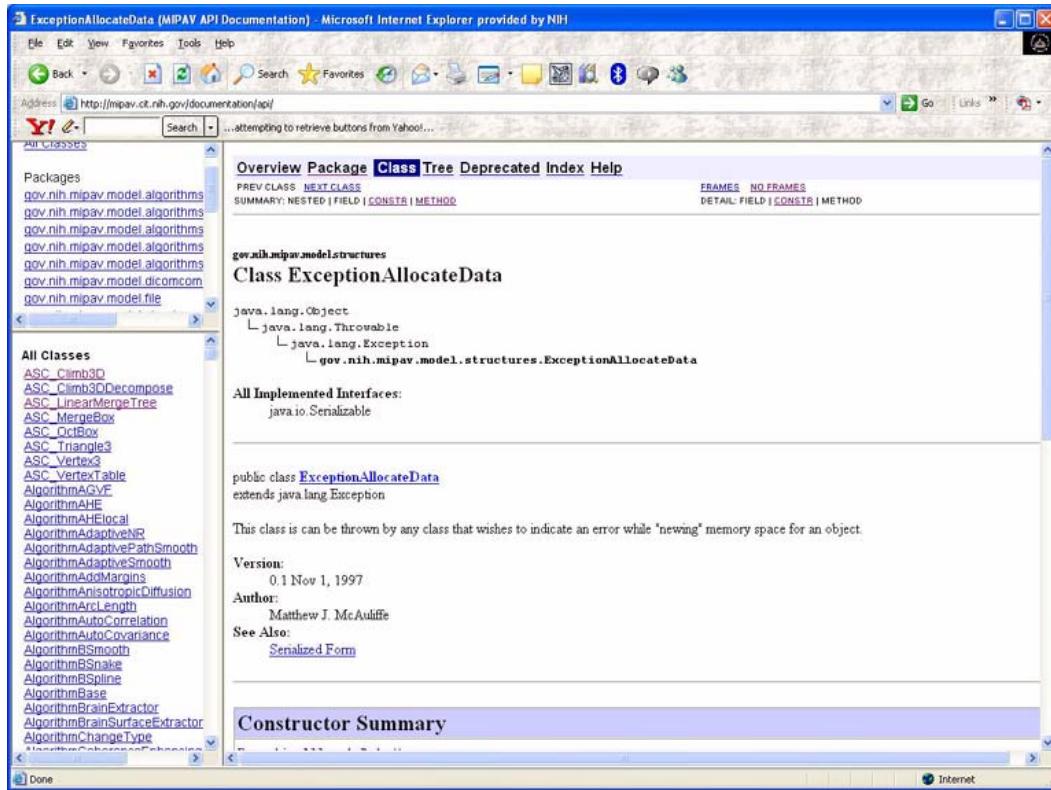
Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.



**Note:** Each serialized or externalized class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. To access this information, go to any serialized class and clicking Serialized Form in the See also section of the class description.

## EXCEPTION PAGE

The Exception page (Figure 304) appears when an exception on the Package page is selected. This page includes a constructor summary and constructor detail.



**Figure 304. The Exception page in the API documentation**

## TREE (CLASS HIERARCHY) PAGE

When you click Tree on the menu, a Tree, or class hierarchy, page (Figure 305 on page 454) appears. This page displays either the class hierarchy for a particular package, or, if you select All Packages, the class hierarchy for all packages.

- If you were viewing the Overview page and then clicked Tree, the class hierarchy for all packages appears on the Tree page.
- If you were viewing a Package, Interface, Class, or Exception page and then clicked Tree, the hierarchy for only that package, which includes the class, interface, and exception hierarchies, appears on the Tree page.

Each hierarchy page contains a list of classes, interfaces, and exceptions (if any). The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

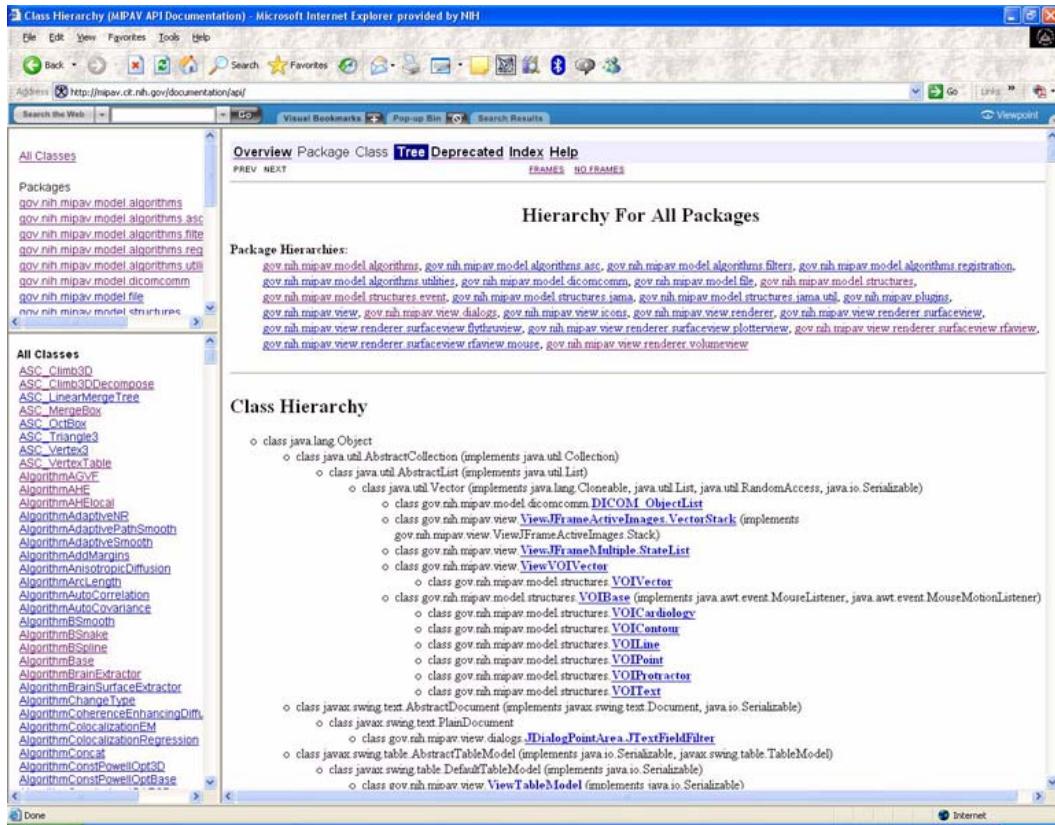


Figure 305. The Tree page in the API documentation

## DEPRECATED API PAGE

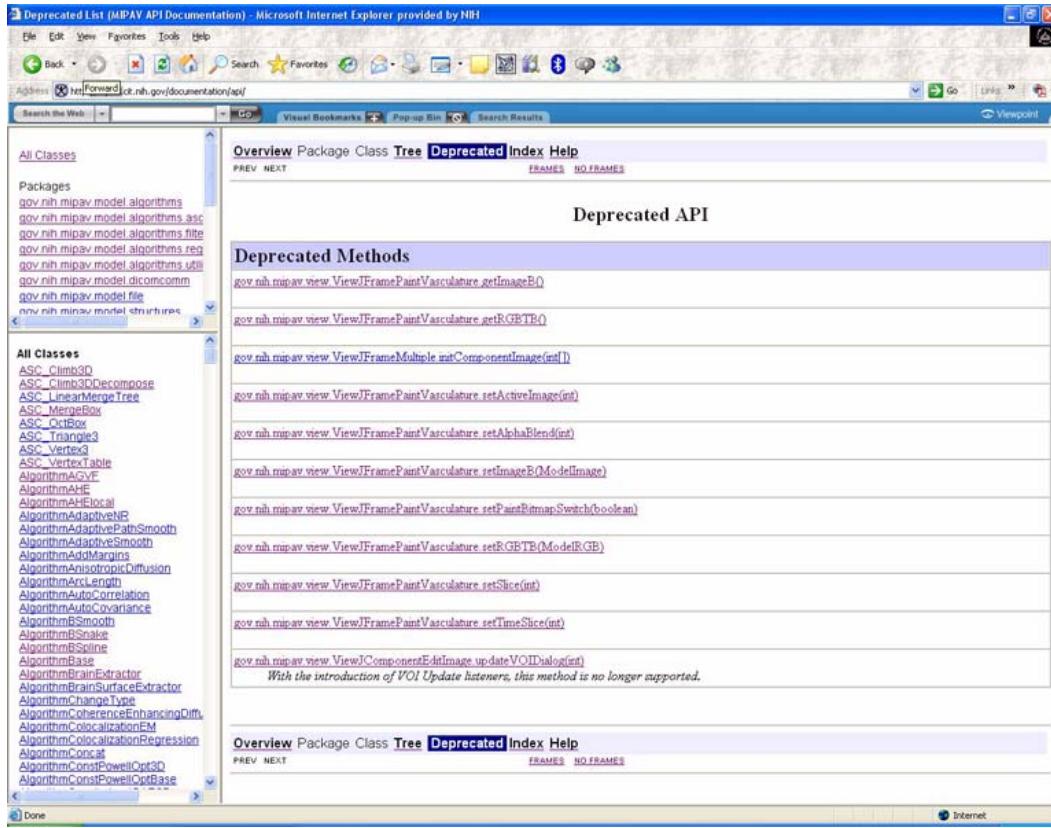
The Deprecated API page (Figure 306) appears when you click Deprecated on the menu. This page lists all of the methods in the API that have been deprecated.



**Recommendation:** A deprecated method is *not recommended* for use, generally due to improvements, and a replacement API is usually given.



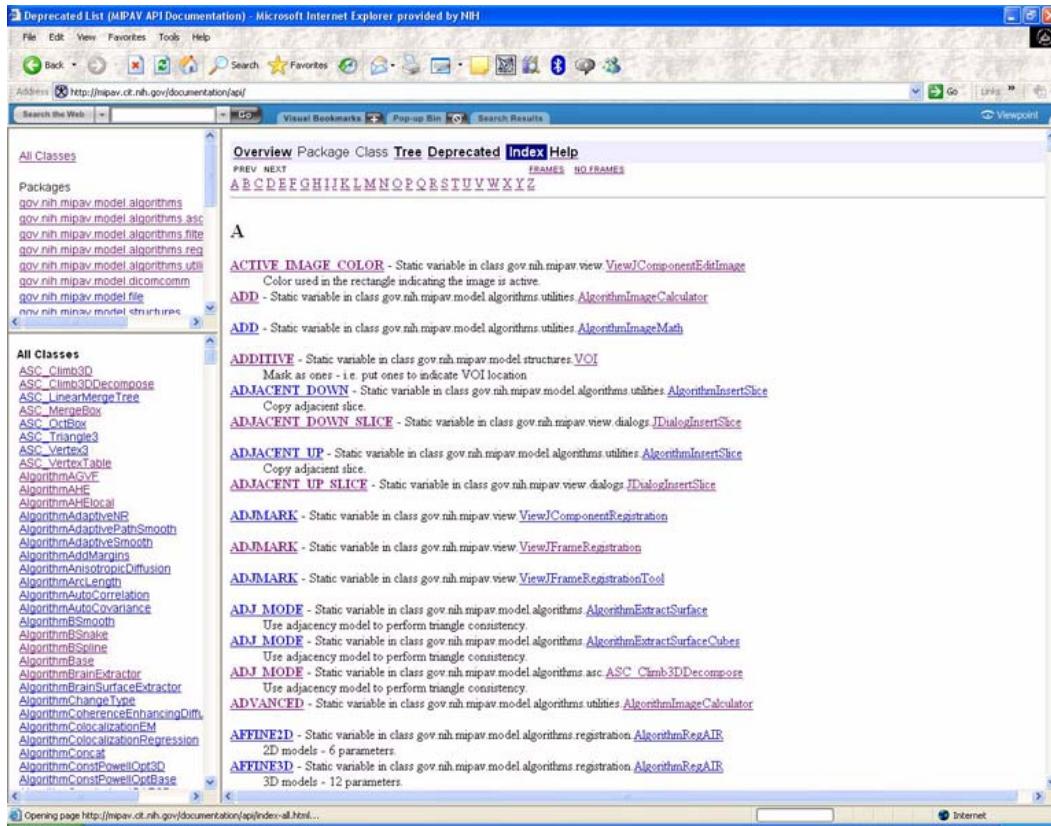
**Note:** Deprecated APIs may be removed in future implementations.



**Figure 306.** The Deprecated page in the API documentation (old methods in the API that are no longer recommended for use)

## INDEX

The Index page (Figure 307) provides an alphabetic list of all classes, interfaces, constructors, methods, and fields with definitions of each. Clicking an entry displays the usage in the product.



**Figure 307. The Index page in the API documentation**

## HELP PAGE

The Help page provides help for using the API documentation.

## To display all interfaces, classes, and exceptions in a package

- 1 Go to <http://mipav.cit.nih.gov/documentation/api/>. The Overview page appears.
- 2 Click one of the packages listed in the:
  - **Frame on the right**—When you click one of the packages listed on this page, the Package page appears in the frame. The Package page displays a list of all interfaces, classes, and exceptions (if any) in the package.
  - **Top frame on the left**—The top frame on the left also lists all of the packages. When you select a package, the bottom frame on the left displays a list of interfaces, classes, and exceptions (if any) in the package.

## To view the methods associated with an interface or with a class

- 1 Go to <http://mipav.cit.nih.gov/documentation/api/>. The Overview page appears.
- 2 Do either of the following:
  - Click one of the packages listed in the frame on the right or in the top frame on the left. The Package page appears in the right frame.
  - Click one of the packages in the top frame on the left. A list of interfaces, classes, and exceptions appear in the bottom frame on the left.
- 3 Do one of the following:
  - Click an interface. The Interface page appears in the right frame.
  - Click a class. The Class page appears in the right frame.
- 4 Scroll down the page, or click METHODS beneath the menu. The Method Summary table appears.
- 5 Click a method. The Method Detail section of the page, which lists a description of the method and its parameters, throws, and returns, appears.
- 6 Click a method. The appropriate section of Java code for MIPAV appears.

## Developing plugin programs

MIPAV provides the following classes for developing plugin programs:

- PlugInAlgorithm.class
- PlugInFile.class
- PlugInView.class

Plugin programs are developed in the same way other Java programs are developed. The high-level steps of creating plugins follow.

- **Step 1, Determining the type of plugin program**—Before you begin to write the code for the plugin, determine the plugin type: algorithm, file, or view.
- **Step 2, Determining which version of Java to use**—Detailed instructions appear in “Step 2, Determining which version of Java to use” on page 459 and Figure 308.
- **Step 3, Writing the source code**—Some lines of code must appear in the source code so that the plugin program interfaces correctly with MIPAV.
- **Step 4, Building and compiling plugin programs**—You should keep back-up copies of the source and compiled files in case you need to update or change plugin programs.
- **Step 5, Installing plugin programs**—This section explains how to install plugin programs.
- **Sample plugin programs**—This section provides samples of plugin programs.



**Note:** This section does not explain how to write a Java program; it explains what must be incorporated in the plugin program so that it correctly interfaces with the MIPAV application.

## Step 1, Determining the type of plugin program

The first step of creating a plugin program is to determine the type you want to create, which depends on its purpose. As mentioned earlier, MIPAV plugin programs can be of the algorithm, file, or view type. However, most users want MIPAV to perform very specific additional functions on images.

Since these functions may not be currently available in MIPAV, users choose to add the functions by developing the algorithm type of plugin program.

## Step 2, Determining which version of Java to use

To avoid compatibility problems when you create a plugin program, use the same version of Java that was used to create MIPAV. To determine which version of Java the latest version of MIPAV uses, select Help > About Java in the MIPAV window. The About System dialog box opens (Figure 308).

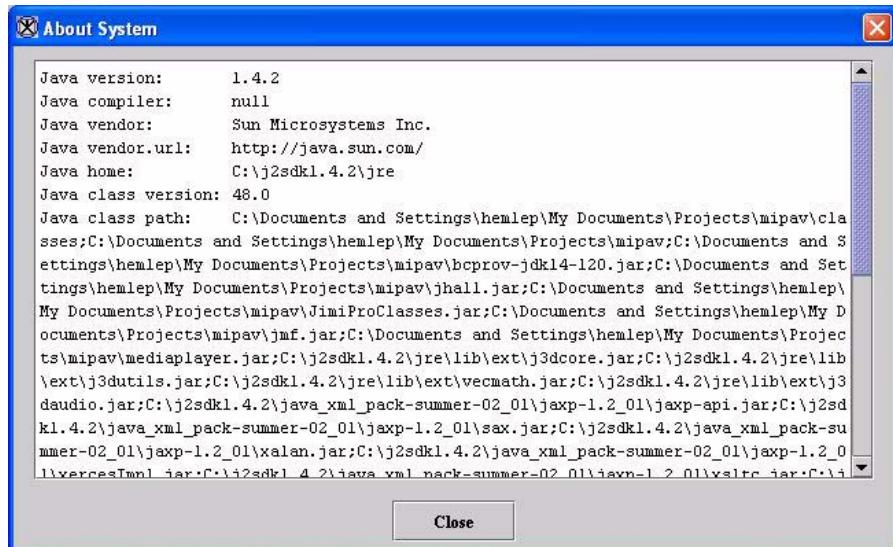


Figure 308. About System dialog box

The first line in the About System dialog box indicates the version of Java that was used to develop MIPAV. To obtain the correct version of Java, go to the following web site:

<http://www.java.sun.com>

## Step 3, Writing the source code



**Note:** In this section, `\$MIPAV` is used to represent the MIPAV user directory, which is the directory where MIPAV is installed. The user directory is indicated in the About System dialog box. In the MIPAV main window, select Help > About Java to view the About System dialog box.

When you develop a plugin for MIPAV, several lines must be present in the code so that it executes properly. Some lines of code should be included in all plugin files. Other lines of code change depending on the plugin type.

### INCLUDING MANDATORY CODE

The next three figures show the mandatory source code needed for creating a file type of plugin (Figure 309), a view type of plugin (Figure 311), and an algorithm type of plugin (Figure 311). The plugins directory of MIPAV (`C:\[$MIPAV]\gov\nih\mipav\plugins`) includes these three files:

- **PlugInFile.java**—Mandatory source code for a file type of plugin
- **PlugInView.java**—Mandatory source code for a view type of plugin
- **PlugInAlgorithm.java**—Mandatory source code for an algorithm type of plugin



**Note:** For readability purposes, keywords in all code reproduced in this chapter appear in bold, and comments appear in green type.

```

1  package gov.nih.mipav.plugins;
2
3  import gov.nih.mipav.view.*;
4
5  import java.awt.*;
6
7  public interface PlugInFile extends PlugIn {
8
9      /**
10      *     run
11      *     @param UI          MIPAV main user interface.
12      */
13      public void run(ViewUserInterface UI);
14  }

```

**Figure 309. Mandatory code for a file type of plugin (PlugInFile.java)**

```

1  package gov.nih.mipav.plugins;
2
3  import gov.nih.mipav.model.structures.*;
4  import gov.nih.mipav.view.*;
5
6  import java.awt.*;
7
8  public interface PlugInView extends PlugIn {
9
10     /**
11     *     run
12     *     @param UI          MIPAV main user interface.
13     *     @param parentFrame frame that displays the MIPAV image.
14     *                         Can be used as a parent frame when building
15     *                         dialogs.
16     *     @param image        model of the MIPAV image.
17     *     @see    ModelImage
18     *     @see    ViewJFrameImage
19     *
20     */
21     public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);
22 }

```

**Figure 310. Mandatory code for a view type of plugin (PlugInView.java)**

```

1  package gov.nih.mipav.plugins;
2
3  import gov.nih.mipav.model.structures.*;
4  import gov.nih.mipav.view.*;
5
6  import java.awt.*;
7
8
9  public interface PlugInAlgorithm extends PlugIn {
10
11     /**
12      * run
13      * @param UI          MIPAV main user interface.
14      * @param parentFrame frame that displays the MIPAV image.
15      *                      Can be used as a parent frame when building
16      *                      dialogs.
17      * @param image        model of the MIPAV image.
18      * @see    ModelImage
19      * @see    ViewJFrameImage
20      *
21     */
22     public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);
23
24
25 }
26

```

**Figure 311.** Mandatory code for an algorithm type of plugin (`PlugInAlgorithm.java`)

## REFERENCING FILES

To reference a class, you must specify it using the Import keyword. For example, line 2 in `PlugInFile.java` imports the view functions (Figure 312).

```
import gov.nih.mipav.view.*;
```

**Figure 312.** Importing the view functions in `PlugInFile.java`

Lines 3, 4, and 6 in the `PlugInView.java` and `PlugInAlgorithm.java` files import the model structures, view functions, and the basic Java package that has GUI functions (Figure 313).

```

import gov.nih.mipav.model.structures.*; // MIPAV package where main
// MIPAV structures are located (e.g., model image)
import gov.nih.mipav.view.*;

import java.awt.*

```

**Figure 313. Importing model structures, view functions, and [java.awt]**

If you reference a class, you must include it in the plugin package so that it can be called from the main file. After you write and compile, you must now install files in the user or home directory:

### **Windows**

c:\Documents and Settings\<user ID>\mipav\plugins

### **Unix**

/user/<user ID>/mipav/plugins

An example of this appears in the first line of Figure 314.

```

package plugins; // added to plugins pkg. so PlugInSampleStub may
// call it.

```

**Figure 314. Example of placing referenced files in the \\$MIPAV\plugins directory**

---

## **LINES OF CODE THAT ARE DEPENDENT ON PLUGIN TYPE**

Two lines of code depend on the type of plugin program being developed:

- Declaration
- Parameters for the run method

### **Declaration**

The declaration used in a plugin depends on the type of plugin being developed. For instance, in line 9 in *PlugInAlgorithm.java* (Figure 311), the word *PlugInAlgorithm* indicates that the plugin is an Algorithm. For File or View types of plugins, simply replace *PlugInAlgorithm* with *PlugInFile* (line 9 in *PlugInFile.java*) (Figure 309) or *PlugInView* (line 9 in *PlugInView.java*) (Figure 310), respectively.

**Table 6. Declarations dependent on type of plugin**

Type of plugin	Declaration
File	<code>public interface PlugInFile extends PlugIn {</code>
View	<code>public interface PlugInView extends PlugIn {</code>
Algorithm	<code>public interface PlugInAlgorithm extends PlugIn {</code>

### Parameters for the run method

The parameters for the run method also depend on the plugin type. Compare the run methods used in PlugInFile.java (Figure 309), PlugInView.java (Figure 310), and PlugInAlgorithm.java (Figure 311).

**Table 7. Parameters for run methods dependent on type of plugin**

Type of plugin	Parameters for the run method
File	<code>public void run(ViewUserInterface UI);</code>
View	<code>public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);</code>
Algorithm	<code>public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);</code>

```

1 package gov.nih.mipav.plugins;
2
3 import gov.nih.mipav.model.structures.*;
4 import gov.nih.mipav.view.*;
5
6 import java.awt.*;
7
8 public interface PlugInAlgorithm extends PlugIn {
9
10 /**
11 * run
12 * @param UI MIPAV main user interface.
13 * @param parentFrame Frame that displays the MIPAV image.
14 * Can be used as a parent frame when building dialogs.
15 * @param image Model of the MIPAV image.
16 * @see ModelImage
17 * @see ViewJFrameImage
18 */
19 public void run(ViewUserInterface UI, Frame parentFrame, ModelImage image);
20
21 }
```

**Figure 315.**

## Step 4, Building and compiling plugin programs

To build a new plugin program for MIPAV, you must first install a build environment, alter the path environment variable, and compile the plugin files.



**Note:** You should keep back-up copies of the source and compiled files in case you need to update or change the plugin.

### INSTALLING A BUILD ENVIRONMENT

- 1** Download and install [Java 2 Platform, Standard Edition, version 1.4 \(J2SE SDK\)](#).
- 2** Download and install [Apache Ant 1.6](#).

### CONFIGURING THE ENVIRONMENT

To configure your environment, you need to add two new variables—*JAVA\_HOME* and *ANT\_HOME*—and update the path variable in your system.

#### On Windows workstations

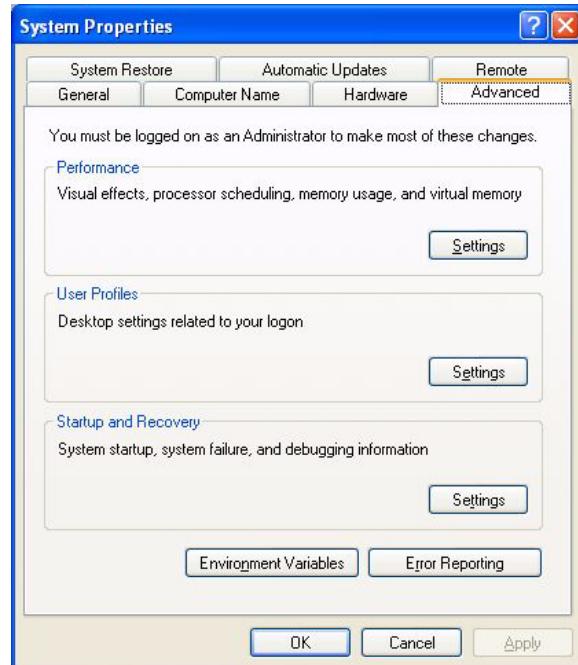
- 1** Click Start > Control Panel. The Control Panel window opens.



- 2** Double-click *System*, the System icon. The System Properties dialog box opens.
- 3** Click Advanced. The Advanced page of the System Properties dialog box (Figure 316 on page 466) appears.
- 4** Click Environment Variables. The Environment Variables dialog box (Figure 317 on page 467) opens.
- 5** Decide whether to add and edit variables in the User variables box or the System variables box based on which users should have access to the Java SDK and Ant.

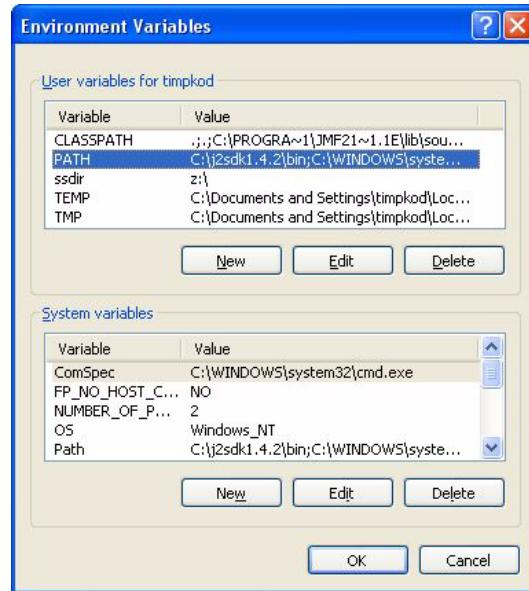


**Note:** Add and edit the variables in the User variables box if you want to limit the build environment to just yourself and no other users. Add and edit the variables in the Systems variables box to make the environment accessible to anyone who uses the workstation.



**Figure 316. System Properties dialog box showing the Advanced page**

- 6** Add the *JAVA\_HOME* variable to your environment by doing the following:
  - a** Click New under either the User variables box or the System variables box. The New User Variable dialog box or the New System Variable dialog box opens as appropriate.
  - b** Type *JAVA\_HOME* in Variable name.
  - c** Type the path for the Java SDK on your computer (e.g., *C:\Program Files\j2sdk1.4.2\_08*) in Variable value.
  - d** Click OK. The *JAVA\_HOME* variable appears in either the User variables box or System variables box as appropriate.



**Figure 317. Environment Variables dialog box**

- 7 Add the *ANT\_HOME* variable to your environment by doing the following:
  - a Click New under either the User variables box or the System variables box. The New User Variable dialog box or the New System Variables dialog box opens as appropriate.
  - b Type *ANT\_HOME* in Variable name.
  - c Type the path for the Ant on your computer (e.g., *C:\Program Files\apache-ant-1.6.3*) in Variable value.
  - d Click OK. The *ANT\_HOME* variable appears in either the User variables box or System variables box as appropriate.
- 8 Update either the *PATH* variable in the User variables box or the *Path* variable in the System variables box by doing the following:
  - a Select the *PATH* variable in the User variables box, or select the *Path* variable in the System variables box.
  - b Click Edit under the User variables box, or click Edit under the System variables box. Either the Edit User Variable dialog box (Figure 318) or the Edit System Variable dialog box opens.



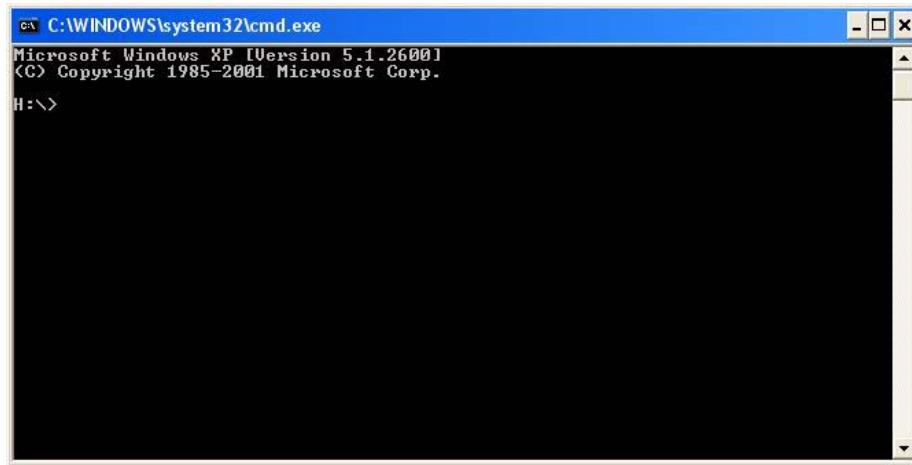
**Figure 318. Edit User Variable dialog box**

- c** Type ;%JAVA\_HOME%\bin;%ANT\_HOME%\bin to the end of the *PATH* variable or to the end of the *Path* variable.
- d** Click OK. The edited variable appears either in the User variables box or the System variables box.



**Recommendation:** Although it is possible to update the path variable in either the User variables box or System variables box, you should add the statement to the same box in which you added the *JAVA\_HOME* and *ANT\_HOME* variables.

- 9** Open a new terminal for the change to take effect by doing the following:
  - a** Click Start > Run. The Run dialog box opens.
  - b** Type **cmd** in Open, and click OK. A terminal window (Figure 319) opens.
- 10** Retrieve the [example Ant build file](#) (Figure 320 on page 471) from the MIPAV web site and place it in the same directory as the plugin .java files you want to compile.
- 11** Alter the *dir.mipav* and *dir.jdk* properties within the *build.xml* to point to the directory where MIPAV and the SDK are installed, respectively.



**Figure 319.** Terminal window

### On Linux or UNIX workstations

Bash users should do the following:

- 1** Edit the file `$HOME/.bash_profile` and add the following lines:

```
ANT_HOME=/path/to/apache-ant-1.6.3
JAVA_HOME=/path/to/j2sdk1.4.2_08
PATH=$PATH:$JAVA_HOME/bin:$ANT_HOME/bin

export ANT_HOME
export JAVA_HOME
export PATH
```

where `ANT_HOME` and `JAVA_HOME` are the paths where each application was installed.

- 2** Retrieve the [example Ant build file](#) from the MIPAV web site, and place it in the same directory as the plugin .java files you want to compile.
- 3** Alter the `dir.mipav` and `dir.jdk` properties within `build.xml` to point to the directory where MIPAV and the SDK are installed, respectively.

---

## COMPIILING THE PLUGIN FILES

- 1** Type **ant compile** on your workstation (e.g., cmd on Windows or xterm on UNIX platforms). The BUILD SUCCESSFUL message should appear at the end of the Ant output.
- 2** Copy the .class files that Ant produced into MIPAV's plugin directory.
  - On Windows platforms:  
C:\Documents and Settings\username\mipav\plugins
  - On UNIX platforms:  
/home/username/mipav/pluginswhere *username* is the name of your account on the system.
- 3** Install the plugin file by selecting Install Plugin in the MIPAV window.

```

1      <?xml version="1.0" encoding="UTF-8"?>
2
3      <!-- build file for MIPAV plugin class -->
4
5      <project basedir=". " default="compile" name="mipav_plugin">
6          <property name="dir.mipav" value="c:\\program files\\mipav\\ "/>
7
8          <property name="dir.jdk" value="c:\\program files\\java\\j2sdk1.4.2_08"/>
9
10         <target name="init">
11             <tstamp/>
12             <path id="build.classpath">
13                 <pathelement path="${dir.mipav}"/>
14
15                 <pathelement location="${dir.mipav}/InsightToolkit/lib/InsightToolkit/
16                     InsightToolkit.jar"/>
17                 <fileset dir="${dir.mipav}">
18                     <filename name="*.jar"/>
19                 </fileset>
20             </path>
21             <property name="build.cp" refid="build.classpath"/>
22         </target>
23
24         <target name="compile" depends="init">
25             <echo>classpath: ${build.cp}</echo>
26
27             <javac  debug="true"
28                   deprecation="true"
29                   description="Builds MIPAV"
30                   verbose="no"
31                   listfiles="yes"
32                   nowarn="no"
33                   fork="true"
34                   memoryInitialSize="220M"
35                   memoryMaximumSize="1000M"
36                   id="mipav build"
37                   source="1.4"
38                   target="1.4"
39                   destdir=". "
40                   srccdir=". "
41                   compiler="modern">
42                     <classpath refid="build.classpath" />
43                 </javac>
44             </target>
45
46             <target name="clean" depends="init">
47                 <delete>
48                     <fileset dir=". ">
49                         <include name="**/*.class"/>
50                     </fileset>
51                 </delete>
52             </target>
53         </project>

```

**Figure 320. The contents of the build.xml file**

## Step 5, Installing plugin programs

Installing simple plugin programs merely copies files into the user's home directory.

### Windows

```
c:\Documents and Settings\<user ID>\mipav\plugins
```

### Unix

```
/user/<user ID>/mipav/plugins
```

You can choose one of two methods for copying the files:

- Use MIPAV's plugin installation tool—in the MIPAV window, select Plugins > Install Plugin.
- Use the operating system's tool for copying the files. This method requires the user to restart MIPAV so that the new plugin appears in thePlugIns menu. When MIPAV starts, it parses the user's home directory and builds the Plugins menu.

The MIPAV installation tool does *not* work for more complex plugins that consist of more complicated package class hierarchy, such as the [Medic Talairach plugin program](#).

## Sample plugin programs

To build plugin programs, three files are typically required:

- **PluginFoo.java**—Provides an interface to MIPAV and the plugin.
- **PluginDialogFoo.java**—Invokes the dialog to get user-supplied parameters; it can be hidden when no parameters are required.
- **PluginAlgorithmFoo.java**—Provides the actual algorithm to be implemented. It can be a mixture of calls to MIPAV's API, C programs, Perl, ITK, etc.

where *Foo* is the name that you supply for the program.

This section includes three sample plugin programs:

- Hello World, a simple program
- PlugInCT\_MD program, a typical program

## HELLO WORLD, A SIMPLE PROGRAM

The program shown in Figure 321 displays a dialog box that says “Hello World.”

```

1 import gov.nih.mipav.plugins.*;      // needed to load PlugInAlgorithm / PlugInView /
2                                     // PlugInFile interface
3 import gov.nih.mipav.view.*;
4 import gov.nih.mipav.model.structures.*;
5 import java.awt.*;
6
7 /**
8 * This is a simple plugin to display a dialog box--hello world
9 * @see PlugInAlgorithm
10 */
11
12 // This is an Algorithm type of PlugIn and therefore must implement PlugInAlgorithm
13 // Implementing the PlugInAlgorithm requires this class to implement the run method
14 // with the correct parameters
15 public class PlugInHello implements PlugInAlgorithm {
16
17 /**
18 * Defines body of run method, which was declared in the interface.
19 * @param UI          User Interface
20 * @param parentFrame parent frame
21 * @param image       Current ModelImage--this is an image already loaded
22 *                   into MIPAV. Can be null.
23 */
24 public void run (ViewUserInterface UI,Frame parentFrame, ModelImage image) {
25
26     JOptionPane.showMessageDialog(null, "Hello World!", "Title-Greetings", JOptionPane.
27             INFORMATION_MESSAGE);
28 }
29 }
```

**Figure 321. Hello World, a simple plugin program**

## PLUGINCT\_MD, A TYPICAL PLUGIN PROGRAM

PlugInCT\_MD is a typical example of a plugin program. It consists of three files:

- **PlugInCT\_MD.java**—Provides an interface to MIPAV and the plugin program.
- **PlugInDialogCT\_MD.java**—Invokes the dialog to get user-supplied parameters.
- **PlugInAlgorithmCT\_MD.java**—Implements the algorithm.

### PlugInCT\_MD.java

The file in Figure 322 provides an interface between MIPAV and PlugInCT\_MD.

```

1   import plugins.PluginDialogCT_MT;      //associated class file
2   import gov.nih.mipav.plugins.*;        //needed to load PlugInAlgorithm / PlugInView /
3   import gov.nih.mipav.view.*;           //PlugInFile interface
4   import gov.nih.mipav.model.structures.*;
5
6
7   import java.awt.*;
8
9 /**
10 * This is a simple plugin for the University of Maryland to simple segment an
11 * imagebased on CT Hounsfield units.
12 *
13 * @see PlugInAlgorithm
14 */
15
16 //This is an Algorithm type of PlugIn, and therefore must implement PlugInAlgorithm
17 //Implementing the PlugInAlgorithm requires this class to implement the run method
18 //with the correct parameters
19 public class PlugInCT_MD implements PlugInAlgorithm {
20
21 /**
22 * Defines body of run method, which was declared in the interface.
23 * @param UI          User Interface
24 * @param parentFrame ParentFrame
25 * @param image       Current ModelImage--this is an image already loaded into
26 *                   MIPAV. Can be null.
27 */
28 public void run (ViewUserInterface UI, Frame parentFrame, ModelImage image){
29
30     if (parentFrame instanceof ViewJFrameImage)
31         new PlugInDialogCT_MD (parentFrame,image);
32
33     else
34         MipavUtil.displayError ("PlugIn CT_MD only runs on an image frame.");
35     }
36 }
37 }
```

**Figure 322. PlugInCT\_MD.java**

### PlugInDialogCT\_MD.java

The file in Figure 323 invokes a dialog box to obtain user-supplied data.

```

1   import gov.nih.mipav.view.*;
2   import gov.nih.mipav.view.dialogs.*;
3   import gov.nih.mipav.model.structures.*;
4   import gov.nih.mipav.model.algorithms.*;

5
6   import java.awt.event.*;
7   import java.awt.*;
8   import java.util.*;

9
10  import javax.swing.*;

11
12
13 /**
14 *
15 * JDIALOGBASE CLASS.
16 *
17 * Note:
18 *
19 * @version July 12, 2002
20 * @author
21 * @see JDIALOGBASE
22 * @see JDIALOGMEDIAN
23 * @see ALGORITHMINTERFACE
24 *
25 * $Logfile: /mipav/src/plugins/PlugInDialogCT_MD.java $
26 * $Revision: 6 $
27 * $Date: 8/05/04 5:44p $
28 *
29 */
30 public class PlugInDialogCT_MD extends JDIALOGBASE implements AlgorithmInterface {
31
32     private PlugInAlgorithmCT_MD ctSegAlgo = null;
33     private ModelImage image; // source image
34     private ModelImage resultImage = null; // result image
35     private ViewUserInterface userInterface;
36
37     private String titles[];
38
39     private float correctionVal;
40     private JTextField fatLValTF;
41     private JTextField fatHValTF;
42     private JTextField ldmLValTF;
43     private JTextField ldmHValTF;
44     private JTextField hdmLValTF;
45     private JTextField hdmHValTF;
46
47     private int fatLVal;
48     private int fatHVal;
49     private int ldmLVal;
50     private int ldmHVal;
51     private int hdmLVal;
52     private int hdmHVal;
53

```

**Figure 323. PlugInDialogCT\_MD.java**

```

54     /**
55      * Creates new dialog for Median filtering using a plugin.
56      * @param parent          Parent frame.
57      * @param im               Source image.
58     */
59
60     public PlugInDialogCT_MD(Frame theParentFrame, ModelImage im) {
61         super(theParentFrame, true);
62         if (im.getType() == ModelImage.BOOLEAN || im.isColorImage()) {
63             MipavUtil.displayError("Source Image must NOT be Boolean or Color");
64             dispose();
65             return;
66         }
67         image = im;
68         userInterface = ((ViewJFrameBase)(parentFrame)).getUserInterface();
69         init();
70     }
71
72     /**
73      * Used primarily for the script to store variables and run the algorithm. No
74      * actual dialog will appear but the set up info and result image will be stored
75      * here.
76      * @param UI   The user interface, needed to create the image frame.
77      * @param imSource image.
78     */
79     public PlugInDialogCT_MD(ViewUserInterface UI, ModelImage im) {
80         super();
81         userInterface = UI;
82         if (im.getType() == ModelImage.BOOLEAN || im.isColorImage()) {
83             MipavUtil.displayError("Source Image must NOT be Boolean or Color");
84             dispose();
85             return;
86         }
87
88         image = im;
89     }
90
91     /**
92      * Sets up the GUI (panels, buttons, etc) and displays it on the screen.
93     */
94     private void init(){
95
96         setForeground(Color.black);
97         setTitle("CT_segmentation");
98
99         JPanel inputPanel = new JPanel(new GridLayout(3, 3));
100        inputPanel.setForeground(Color.black);
101        inputPanel.setBorder(buildTitledBorder("Input parameters"));
102
103        JLabel labelFat = new JLabel("Fat thresholds: ");
104        labelFat.setForeground(Color.black);
105        labelFat.setFont(serif12);
106        inputPanel.add(labelFat);
107

```

**Figure 323. PlugInDialogCT\_MD.java (continued)**

```

108         fatLValTF = new JTextField();
109         fatLValTF.setText("-190");
110         fatLValTF.setFont(serif12);
111         inputPanel.add(fatLValTF);
112
113         fatHValTF = new JTextField();
114         fatHValTF.setText("-30");
115         fatHValTF.setFont(serif12);
116         inputPanel.add(fatHValTF);
117
118         JLabel labelLDM = new JLabel("Low density muscle thresholds: ");
119         labelLDM.setForeground(Color.black);
120         labelLDM.setFont(serif12);
121         inputPanel.add(labelLDM);
122
123         ldmLValTF = new JTextField();
124         ldmLValTF.setText("0");
125         ldmLValTF.setFont(serif12);
126         inputPanel.add(ldmLValTF);
127
128         ldmHValTF = new JTextField();
129         ldmHValTF.setText("30");
130         ldmHValTF.setFont(serif12);
131         inputPanel.add(ldmHValTF);
132
133         JLabel labelHDM = new JLabel("High density muscle thresholds: ");
134         labelHDM.setForeground(Color.black);
135         labelHDM.setFont(serif12);
136         inputPanel.add(labelHDM);
137
138         hdmLValTF = new JTextField();
139         hdmLValTF.setText("31");
140         hdmLValTF.setFont(serif12);
141         inputPanel.add(hdmLValTF);
142
143         hdmHValTF = new JTextField();
144         hdmHValTF.setText("100");
145         hdmHValTF.setFont(serif12);
146         inputPanel.add(hdmHValTF);
147
148         getContentPane().add(inputPanel, BorderLayout.CENTER);
149
150         // Build the Panel that holds the OK and CANCEL Buttons
151         JPanel OKCancelPanel = new JPanel();
152
153         // size and place the OK button
154         buildOKButton();
155         OKCancelPanel.add(OKButton, BorderLayout.WEST);
156         // size and place the CANCEL button
157         buildCancelButton();
158         OKCancelPanel.add(cancelButton, BorderLayout.EAST);
159         getContentPane().add(OKCancelPanel, BorderLayout.SOUTH);

```

**Figure 323. PluginDialogCT\_MD.java (continued)**

```

160         pack();
161         setVisible(true);
162         setResizable(false);
163         System.gc();
164     } // end init()
165
166
167     /**
168      * Accessor that returns the image.
169      * @return          The result image.
170     */
171    public ModelImage getResultImage(){return resultImage;}
172
173
174
175     /**
176      * Accessor that sets the correction value
177      * @param num Value to set iterations to (should be between 1 and 20).
178     */
179    public void setCorrectionValue(float num){correctionVal = num;}
180
181 //*****
182 //***** Event Processing *****
183 //*****
184
185     /**
186      * Closes dialog box when the OK button is pressed and calls the algorithm.
187      * @param event      Event that triggers function.
188     */
189    public void actionPerformed(ActionEvent event) {
190        String command = event.getActionCommand();
191
192        if (command.equals("OK")) {
193            if (setVariables()) {
194                callAlgorithm();
195            }
196        }
197        else if (command.equals("Script")) {
198            callAlgorithm();
199        }
200        else if (command.equals("Cancel")) {
201            dispose();
202        }
203    }
204
205 //*****
206 //***** Algorithm Events *****
207 //*****
208
209 /**
210  * This method is required if the AlgorithmPerformed interface is implemented.
211  * It is called by the algorithm when it has completed or failed to complete,
212  * so that the dialog can be display the result image and/or clean up.
213  * @param algorithm  Algorithm that caused the event.
214 */
215    public void algorithmPerformed(AlgorithmBase algorithm) {

```

**Figure 323. PluginDialogCT\_MD.java (continued)**

```

216     ViewJFrameImage imageFrame = null;
217     if ( algorithm instanceof PlugInAlgorithmCT_MD) {
218         image.clearMask();
219         if(ctSegAlgo.isCompleted() == true && resultImage != null) {
220             //The algorithm has completed and produced a new image to be displayed.
221
222             updateFileInfo(image, resultImage);
223             resultImage.clearMask();
224             try {
225                 //resultImage.setImageName("Median: "+image.getImageName());
226
227                 int dimExtentsLUT[] = new int[2];
228                 dimExtentsLUT[0] = 4;
229                 dimExtentsLUT[1] = 256;
230                 ModellLUT LUTa = new ModellLUT(ModelLUT.COOLHOT, 256, dimExtentsLUT);
231                 imageFrame = new ViewJFrameImage(resultImage, LUTa, new Dimension(610,200),
232                                         userInterface);
233             }
234             catch (OutOfMemoryError error){
235                 System.gc();
236                 MipavUtil.displayError("Out of memory: unable to open new frame");
237             }
238         }
239         else if (resultImage == null) {
240             // These next lines set the titles in all frames where the source image
241             // is displayed to image name so as to indicate that the image is now
242             // unlocked! The image frames are enabled and then registered to the
243             // userinterface.
244             Vector imageFrames = image.getImageFrameVector();
245             for (int i = 0; i < imageFrames.size(); i++) {
246                 ((Frame)(imageFrames.elementAt(i))).setTitle(titles[i]);
247                 ((Frame)(imageFrames.elementAt(i))).setEnabled(true);
248                 if ( ((Frame)(imageFrames.elementAt(i))) != parentFrame) {
249                     userInterface.registerFrame((Frame)(imageFrames.elementAt(i)));
250                 }
251             }
252             if (parentFrame != null) userInterface.registerFrame(parentFrame);
253             image.notifyImageDisplayListeners(null, true);
254         }
255         else if (resultImage != null){
256             //algorithm failed but result image still has garbage
257             resultImage.disposeLocal(); // clean up memory
258             resultImage = null;
259             System.gc();
260         }
261     }
262     if (ctSegAlgo.isCompleted() == true) {
263         if (userInterface.isScriptRecording()) {
264             userInterface.getScriptDialog().append("Flow " +
265             userInterface.getScriptDialog().getVar(image.getImageName()) + " "
266             + correctionVal + "\n");
267         }
268     }
269     dispose();
270

```

**Figure 323. PlugInDialogCT\_MD.java (continued)**

```

271     } // end AlgorithmPerformed()
272
273
274     /**
275      * Use the GUI results to set up the variables needed to run the algorithm.
276      * @return      <code>true</code> if parameters set successfully, <code>false
277      *      </code> otherwise.
278     */
279     private boolean setVariables() {
280         String tmpStr;
281
282
283         // verify iteration is within bounds
284         tmpStr = fatLValTF.getText();
285         if ( testParameter(tmpStr, -4000, 4000) ){
286             fatLVal = Integer.valueOf(tmpStr).intValue();
287         }
288         else{
289             fatLValTF.requestFocus();
290             fatLValTF.selectAll();
291             return false;
292         }
293
294         tmpStr = fatHValTF.getText();
295         if ( testParameter(tmpStr, -4000, 4000) ){
296             fatHVal = Integer.valueOf(tmpStr).intValue();
297         }
298         else{
299             fatHValTF.requestFocus();
300             fatHValTF.selectAll();
301             return false;
302         }
303
304         tmpStr = ldmLValTF.getText();
305         if ( testParameter(tmpStr, -4000, 4000) ){
306             ldmLVal = Integer.valueOf(tmpStr).intValue();
307         }
308         else{
309             ldmLValTF.requestFocus();
310             ldmLValTF.selectAll();
311             return false;
312         }
313
314         tmpStr = ldmHValTF.getText();
315         if ( testParameter(tmpStr, -4000, 4000) ){
316             ldmHVal = Integer.valueOf(tmpStr).intValue();
317         }
318         else{
319             ldmHValTF.requestFocus();
320             ldmHValTF.selectAll();
321             return false;
322         }
323
324

```

**Figure 323. PluginDialogCT\_MD.java (continued)**

```

325     tmpStr = hdmLValTF.getText();
326     if ( testParameter(tmpStr, -4000, 4000) ){
327         hdmLVal = Integer.valueOf(tmpStr).intValue();
328     }
329     else{
330         hdmLValTF.requestFocus();
331         hdmLValTF.selectAll();
332         return false;
333     }
334
335     tmpStr = hdmHValTF.getText();
336     if ( testParameter(tmpStr, -4000, 4000) ){
337         hdmHVal = Integer.valueOf(tmpStr).intValue();
338     }
339     else{
340         hdmHValTF.requestFocus();
341         hdmHValTF.selectAll();
342         return false;
343     }
344
345     return true;
346 } // end setVariables()
347
348 /**
349 *      Once all the necessary variables are set, call the Gaussian Blur
350 *      algorithm based on what type of image this is and whether or not there
351 *      is a separate destination image.
352 */
353 private void callAlgorithm() {
354     String name = makeImageName(image.getimageName(), "_CTseg");
355
356     // stuff to do when working on 2-D images.
357     if (image.getNDims() == 2 ) { // source image is 2D
358         int destExtents[] = new int[2];
359         destExtents[0] = image.getExtents()[0]; // X dim
360         destExtents[1] = image.getExtents()[1]; // Y dim
361
362         try{
363             // Make result image of Ubyte type
364             resultImage = new ModelImage(ModelStorageBase.UBYTE, destExtents, name,
365                                         userInterface);
366
367             // Make algorithm
368             boolean entireFlag = true;
369
370             //ctSegAlgo = new PlugInAlgorithmFlowWrapFix(resultImage, image, iters,
371             // kernelSize, kernelShape, stdDev, regionFlag);
372             ctSegAlgo = new PlugInAlgorithmCT_MD(resultImage, image);
373
374             System.out.println("Dialog fatL = " + fatLVal + " fatH = " + fatHVal);
375             ctSegAlgo.fatL = fatLVal;
376             ctSegAlgo.fatH = fatHVal;
377             ctSegAlgo.ldmL = ldmLVal;
378             ctSegAlgo.ldmH = ldmHVal;

```

**Figure 323. *PlugInDialogCT\_MD.java* (continued)**

```

379             ctSegAlgo.hdmL = hdmLVal;
380             ctSegAlgo.hdmH = hdmHVal;
381
382
383
384         // This is very important. Adding this object as a listener allows the
385         // algorithm to notify this object when it has completed or failed. See
386         // algorithm performed event.
387         // This is made possible by implementing AlgorithmmedPerformed interface
388         ctSegAlgo.addListener(this);
389         setVisible(false); // Hide dialog
390
391     if (runInSeparateThread) {
392         // Start the thread as a low priority because we wish to still have
393         // user interface work fast.
394         if (ctSegAlgo.startMethod(Thread.MIN_PRIORITY) == false){
395             MipavUtil.displayError("A thread is already running on this object");
396         }
397     }
398     else {
399         ctSegAlgo.run();
400     }
401 }
402 catch (OutOfMemoryError x){
403     MipavUtil.displayError("Dialog median: unable to allocate enough memory");
404     if (resultImage != null){
405         resultImage.disposeLocal(); // Clean up memory of result image
406         resultImage = null;
407     }
408     return;
409 }
410 }
411 else if (image.getNDims() == 3 ) {
412     int destExtents[] = new int[3];
413     destExtents[0] = image.getExtents()[0];
414     destExtents[1] = image.getExtents()[1];
415     destExtents[2] = image.getExtents()[2];
416
417     try{
418         // Make result image of float type
419         resultImage = new ModelImage(ModelStorageBase.UBYTE, destExtents, name,
420                                     userInterface);
421         boolean entireFlag = true;
422
423         ctSegAlgo = new PlugInAlgorithmCT_MD(resultImage, image);
424         ctSegAlgo.fatL = fatLVal;
425         ctSegAlgo.fatH = fatHVal;
426         ctSegAlgo.ldmL = ldmLVal;
427         ctSegAlgo.ldmH = ldmHVal;
428         ctSegAlgo.hdmL = hdmLVal;
429         ctSegAlgo.hdmH = hdmHVal;
430

```

**Figure 323. *PlugInDialogCT\_MD.java* (continued)**

```

431             // This is very important. Adding this object as a listener allows the
432             // algorithm to notify this object when it has completed or failed.
433             // See algorithm performed event. This is made possible by implementing
434             // AlgorithmPerformed interface
435             ctSegAlgo.addListener(this);
436             setVisible(false);           // Hide dialog
437
438             if (runInSeparateThread) {
439                 // Start the thread as a low priority because we wish to still have
440                 // user interface work fast.
441                 if (ctSegAlgo.startMethod(Thread.MIN_PRIORITY) == false){
442                     MipavUtil.displayError("A thread is already running on this object");
443                 }
444             }
445             else {
446                 ctSegAlgo.run();
447             }
448         }
449         catch (OutOfMemoryError x){
450             MipavUtil.displayError("Dialog median: unable to allocate enough memory");
451             if (resultImage != null){
452                 resultImage.disposeLocal();      // Clean up image memory
453                 resultImage = null;
454             }
455             return;
456         }
457     }
458 } // end callAlgorithm()
459
460 }
```

**Figure 323. PluginDialogCT\_MD.java (continued)**

## PlugInAlgorithmCT\_MD.java

Figure 324 shows the content of PlugInAlgorithmCT\_MD.java.

```

1  import gov.nih.mipav.model.algorithms.*;
2  import gov.nih.mipav.model.structures.*;
3  import gov.nih.mipav.view.*;
4
5  import java.io.*;
6  import java.util.*;
7
8
9  /**
10 *
11 *   This shows how to extend the AlgorithmBase class.
12 *
13 *   Supports the segmentation
14 *   CT scans:
15 *       Fat:           -190 to -30
16 *       Low density muscle:    0 to 30
17 *       High density muscle:  31 to 100
18 *       If you have any questions, please drop me a line.
19 * ====
20 * Matthew J. Delmonico, MS, MPH
21 * Graduate Research Assistant, Exercise Physiology
22 * 2132 HHP Building
23 * University of Maryland
24 * College Park, MD 20742
25 * (301) 405-2569
26 * (301) 793-0567 (cell)
27 *
28 * @version July 12, 2002
29 * @author
30 * @see AlgorithmBase
31 *
32 * $Logfile: /mipav/src/plugins/PlugInAlgorithmCT_MD.java $
33 * $Revision: 10 $
34 * $Date: 10/13/04 1:09p $
35 *
36 */
37 public class PlugInAlgorithmCT_MD extends AlgorithmBase {
38
39
40     private boolean      entireImage = true;
41
42     public int           fatL     = -190;
43     public int           fatH     = -30;
44
45     public int           ldmL     = 0;
46     public int           ldmH     = 30;
47
48     public int           hdmL     = 31;
49     public int           hdmH     = 100;
50
51
52     /**
53     * Constructor for 3D images in which changes are placed in a predetermined
54     * destination image.
55     * @param destImg      Image model where result image is to stored.
56     * @param srcImg       Source image model.
57

```

```

58     */
59     public PlugInAlgorithmCT_MD(ModelImage destImg, ModelImage srcImg) {
60         super(destImg, srcImg);
61     }
62
63     /**
64      * Prepares this class for destruction.
65     */
66     public void finalize(){
67         destImage    = null;
68         srcImage    = null;
69         super.finalize();
70     }
71
72
73
74     /**
75      * Starts the algorithm.
76     */
77     public void run() {
78
79         if (srcImage == null) {
80             displayError("Source Image is null");
81             notifyListeners(this);
82             return;
83         }
84         if (destImage == null) {
85             displayError("Source Image is null");
86             notifyListeners(this);
87             return;
88         }
89
90         // start the timer to compute the elapsed time
91         setStartTime();
92
93
94         if (destImage != null){      // if there exists a destination image
95             if (srcImage.getNDims() == 2){
96                 calcStoreInDest2D();
97             }
98             else if (srcImage.getNDims() > 2) {
99                 calcStoreInDest3D();
100            }
101        }
102
103        // compute the elapsed time
104        computeElapsedTime();
105        notifyListeners(this);
106    }
107
108    /**
109     * This function produces a new image that has been median filtered and places
110     * filtered image in the destination image.
111     */
112    private void calcStoreInDest2D(){
113
114        int length;           // total number of data-elements (pixels) in image
115        float buffer[];       // data-buffer (for pixel data) which is the "heart"
116                                         // of the image
117

```

```

118         try {
119             // image length is length in 2 dims
120             length = srcImage.getExtents()[0] * srcImage.getExtents()[1];
121             buffer      = new float[length];
122             srcImage.exportData(0,length, buffer); // locks and releases lock
123         }
124         catch (IOException error) {
125             buffer = null;
126             errorCleanUp("Algorithm CT_MD reports: source image locked", true);
127             return;
128         }
129         catch (OutOfMemoryError e){
130             buffer = null;
131             errorCleanUp("Algorithm CT_MD reports: out of memory", true);
132             return;
133         }
134
135         int mod = length/100; // mod is 1 percent of length
136         initProgressBar();
137
138         // Fat: -190 to -30
139         // Low density muscle: 0 to 30
140         // High density muscle: 31 to 100
141         BitSet mask = null;
142         if (srcImage.getVOIs().size() > 0 ) {
143             mask = srcImage.generateVOIMask();
144             entireImage = false;
145         }
146
147         int fat      = 0;
148         int ldMuscle = 0;
149         int hdMuscle = 0;
150         for (int i = 0; i < length && !threadStopped; i++){
151             if (isProgressBarVisible() && (i)%mod==0)
152                 progressBar.setValue(Math.round((float)i)/(length-1) * 100));
153
154             if (entireImage == true || mask.get(i) ) {
155                 if( buffer[i] >= fatL && buffer[i] <= fatH ) {
156                     destImage.set(i, 20);
157                     fat++;
158                 }
159                 else if( buffer[i] >= ldmL && buffer[i] <= ldMh ) {
160                     destImage.set(i, 40);
161                     ldMuscle++;
162                 }
163                 else if( buffer[i] >= hdmL && buffer[i] <= hdmH ) {
164                     destImage.set(i, 60);
165                     hdMuscle++;
166                 }
167                 else {
168                     destImage.set(i, 0);
169                     //buffer[i] = (float)srcImage.getMin();
170                 }
171             }
172         }
173     }
174
175

```

**Figure 324. PluginAlgorithmCT\_MD.java**

```

176         //destImage.releaseLock();
177
178         if (threadStopped) {
179             finalize();
180             return;
181         }
182
183         float area = srcImage.getFileInfo()[0].getResolutions()[0] *
184             srcImage.getFileInfo()[0].getResolutions()[1];
185
186         destImage.getUserInterface().getMessageFrame().append("Number of Fat pixels = " +
187             fat , ViewJFrameMessage.DATA );
188         destImage.getUserInterface().getMessageFrame().append(" Area = " + (fat*area) +
189             " mm^2\n", ViewJFrameMessage.DATA );
190
191         destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
192             ldMuscle , ViewJFrameMessage.DATA );
193         destImage.getUserInterface().getMessageFrame().append(" Area = " + (ldMuscle*area) +
194             " mm^2\n", ViewJFrameMessage.DATA );
195
196         destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels = " +
197             hdMuscle , ViewJFrameMessage.DATA );
198         destImage.getUserInterface().getMessageFrame().append(" Area = " + (hdMuscle*area) +
199             " mm^2\n", ViewJFrameMessage.DATA );
200
201         destImage.calcMinMax();
202         setCompleted(true);
203     }
204
205     /**
206      * This function produces a new volume image that has been median filtered.
207      * Image can be filtered by filtering each slice individually, or by filtering
208      * using a kernel-volume.
209      */
210     private void calcStoreInDest3D(){
211
212         int totLength, imgLength;
213         float buffer[];
214
215         float vol = srcImage.getFileInfo()[0].getResolutions()[0] *
216             srcImage.getFileInfo()[0].getResolutions()[1] *
217             srcImage.getFileInfo()[0].getResolutions()[2];
218
219         try {
220             // image totLength is totLength in 3 dims
221             imgLength = srcImage.getSliceSize();
222             totLength = srcImage.getSliceSize() * srcImage.getExtents()[2];
223             buffer = new float[totLength];
224             srcImage.exportData(0,totLength, buffer); // locks and releases lock
225             buildProgressBar(srcImage.getImageName(), "Processing image ...", 0, 100);
226         }
227
228         catch (IOException error) {
229             buffer = null;
230             errorCleanUp("Algorithm CT_MD: source image locked", true);
231             return;
232         }

```

**Figure 324. PluginAlgorithmCT\_MD.java**

```

233         catch (OutOfMemoryError e){
234             buffer = null;
235             errorCleanUp("Algorithm CT_MD: Out of memory creating process buffer", true);
236             return;
237         }
238
239         int totFat      = 0;
240         int totLdMuscle = 0;
241         int totHdMuscle = 0;
242         initProgressBar();
243
244         for (int i = 0; i < srcImage.getExtents()[2] && !threadStopped; i++){
245             int fat      = 0;
246             int ldMuscle = 0;
247             int hdMuscle = 0;
248
249             if ( isProgressBarVisible() )
250                 progressBar.setValue(Math.round((float)(i)/(srcImage.getExtents()[2]-1) *
251                                         100));
252
253             for (int j = 0; j < imgLength && !threadStopped; j++){
254                 //System.out.println(" j = " + j);
255                 int index = i*imgLength+j;
256                 if( buffer[index] >= fatL && buffer[index] <= fatH ) {
257                     destImage.set(index, 60);
258                     totFat++;
259                     fat++;
260                 }
261                 else if( buffer[index] >= ldmL && buffer[index] <= ldmH ) {
262                     destImage.set(index, 120);
263                     totLdMuscle++;
264                     ldMuscle++;
265                 }
266                 else if( buffer[index] >= hdmL && buffer[index] <= hdmH ) {
267                     destImage.set(index, 200);
268                     totHdMuscle++;
269                     hdMuscle++;
270                 }
271                 else {
272                     destImage.set(index, 0);
273                     //buffer[i] = -1024;
274                 }
275             }
276             destImage.getUserInterface().getMessageFrame().append("\n\n ***** Slice
277             " + i + " totals *****\n",
278             ViewJFrameMessage.DATA);
279             destImage.getUserInterface().getMessageFrame().append("Number of fat pixels = " +
280                                         fat , ViewJFrameMessage.DATA );
281             destImage.getUserInterface().getMessageFrame().append(" Volume = " + (fat*vol) +
282                                         " mm^3\n", ViewJFrameMessage.DATA );
283
284
285             destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
286                                         ldMuscle , ViewJFrameMessage.DATA );
287             destImage.getUserInterface().getMessageFrame().append(" Volume = " +
288                                         (ldMuscle*vol) + " mm^3\n", ViewJFrameMessage.DATA );
289

```

**Figure 324. PluginAlgorithmCT\_MD.java**

```

290         destImage.getUserInterface().getMessageFrame().append("Number of HDM pixels
291             = " + hdMuscle , ViewJFrameMessage.DATA );
292         destImage.getUserInterface().getMessageFrame().append(" Volume = " +
293             (hdMuscle*vol) + " mm^3\n", ViewJFrameMessage.DATA );
294     }
295
296     destImage.releaseLock();
297
298     if (threadStopped) {
299         finalize();
300         return;
301     }
302
303
304
305     destImage.getUserInterface().getMessageFrame().append("\n *****\n",
306         Totals *****\n",
307         ViewJFrameMessage.DATA);
308     destImage.getUserInterface().getMessageFrame().append("Number of totFat pixels = " +
309             totFat , ViewJFrameMessage.DATA );
310     destImage.getUserInterface().getMessageFrame().append(" Volume = " + (totFat*vol) +
311             " mm^3\n", ViewJFrameMessage.DATA );
312
313     destImage.getUserInterface().getMessageFrame().append("Number of LDM pixels = " +
314             totLdMuscle , ViewJFrameMessage.DATA );
315     destImage.getUserInterface().getMessageFrame().append(" Volume = " + (totLdMuscle*vol)
316             + " mm^3\n", ViewJFrameMessage.DATA );
317
318     destImage.getUserInterface().getMessageFrame().append("Number of HdM pixels = " +
319             totHdMuscle , ViewJFrameMessage.DATA );
320     destImage.getUserInterface().getMessageFrame().append(" Volume = " + (totHdMuscle*vol)
321             + " mm^3\n", ViewJFrameMessage.DATA );
322
323     destImage.calcMinMax();
324     progressBar.dispose();
325     setCompleted(true);
326 }
327
328 }
```

**Figure 324. PlugInAlgorithmCT\_MD.java**

## PlugInAlgorithmMedian

The source code for the plugin program, *PlugInAlgorithmMedian.java*, which appears on the following pages, is an example of an algorithm type of plugin. This plugin program runs a median filter on an image, using its own dialog box and implementation of the median filter.




---

**Note:** For ease of reading, comment lines in the source file appear in green type, and keywords appear in bold type.

---

```

1 // By leaving out the package keyword in this class, it is therefore in the default
2 // package
3 // for the application.
4
5 import gov.nih.mipav.model.algorithms.*;
6 import gov.nih.mipav.model.structures.*;
7 import gov.nih.mipav.view.*;
8
9 import java.io.*;
10 import java.util.*;
11 import java.awt.*;
12
13 /**
14 * Example of a plugin implementation of the median filter.
15 * This class creates the algorithm that runs on the image.
16 * This shows how to extend the AlgorithmBase class.
17 *
18 * Note: The median algorithm is already implemented in
19 *       the MIPAV/IASO software.
20 *
21 * @version July 12, 2002
22 * @see Algorithms
23 * @see AlgorithmMedian
24 *
25 * $Logfile: /mipav/src/plugins/PlugInAlgorithmMedian.java $
26 * $Revision: 7 $
27 * $Date: 3/16/05 3:36p $
28 *
29 */
30 public class PlugInAlgorithmMedian extends AlgorithmBase {
31
32     private static final int     SQUARE_KERNEL      = 0;    //square kernel
33     private static final int     CUBE_KERNEL        = 0;
34     private static final int     CROSS_KERNEL       = 1;    // cross
35     private static final int     AXIAL_KERNEL       = 1;    //
36     private static final int     X_KERNEL           = 2;    // X-shaped kernel, from 1
37                                         // corner to opposite corner
38     private static final int     HORZ_KERNEL        = 3;    // horizontal (2D only)
39     private static final int     VERT_KERNEL        = 4;    // vertical (2D only)
40
41     private BitSet               mask = null;
42
43     private int                  iterations;          // number of times to filter
44                                         // the image.
45
46     private int                  kernelSize;         // dimension of the kernel
47                                         // (i.e., 5 = 5x5, 7 = 7x7,
48                                         // 9 = 9x9, etc.)
49     private int                  kernelShape;        // user-selectable shape of
50                                         // the region for neighbor-
51                                         // selection
52     private boolean              entireImage;        // true means apply to entire
53                                         // image, false only region
54     private byte[]                kernel;            // mask to determine the
55                                         // region of pixels used in a
56                                         // median filter
57     private float                 stdDevLimit;        // compute median value of
58                                         // pixel if pixel magnitude is
59                                         // outside this fraction of

```

```

60      private boolean           sliceFiltering;          // the standard deviation
61      private int              currentSlice = 0; // do all filtering slice-by-
62      private int              numberOfSlices;
63      private int              halfK;
64      private int              kernelCenter;
65      private float[]          kernelMask;
66      private int              maskCenter;
67
68
69
70
71
72
73
74      private boolean           isColorImage = false; // indicates the image being
75                                         // messed with is a color
76                                         // image
77      private int              valuesPerPixel=1; // number of elements in a
78                                         // pixel. Monochrome = 1,
79                                         // Color = 4. (a, R, G, B)
80      private boolean           rChannel = true; // if T, filter the red
81                                         // channel
82      private boolean           gChannel = true; // the green channel
83      private boolean           bChannel = true; // the blue channel
84
85 /**
86 *          Constructor for 3D images in which changes are placed in a predetermined
87 *          destination image.
88 * @param destImg           Image model where result image is to stored.
89 * @param srcImg            Source image model.
90 * @param iters              Number of iterations of the median filter.
91 * @param kSize              Kernel size: dimension of the kernel (i.e., 5 = 5x5,
92 *                         7 = 7x7, 9 = 9x9, etc.).
93 * @param kShape             Kernel shape: element neighbors to include when finding
94 *                         the median.
95 * @param stdDev             Inner-bounds by which to process pixels (pixel values
96 *                         outside this bound will be median filtered).
97 * @param sliceBySlice       Each slice in a volume image is to be filtered separately
98 *                         (when true), else the volume will use a kernel with 3
99 *                         dimensions.
100 * @param maskFlag          Flag that indicates that the median filtering will be
101 *                         performed for the whole image if equal to true.
102 */
103     public PlugInAlgorithmMedian(ModelImage destImg, ModelImage srcImg, int iters,
104                               int kSize, int kShape, float stdDev, boolean sliceBySlice,
105                               boolean maskFlag) {
106
107     super(destImg, srcImg);
108     if ( srcImg.isColorImage() ) {
109         isColorImage = true;
110         valuesPerPixel = 4;
111     }
112     // else, already false
113     entireImage  = maskFlag;
114     iterations   = iters;
115     kernelSize   = kSize;           // dimension of the kernel
116     kernelShape  = kShape;         // set up the mask (kernel) used to
117                                         // filter
118     stdDevLimit  = stdDev;        // inside magnitude bounds of pixel
119                                         // value to adjust

```

```

120         sliceFiltering = sliceBySlice;
121         numberOfSlices = srcImage.getExtents()[2];
122         makeKernel();
123     }
124
125 /**
126 * Constructor for 2D images in which changes are placed in a predetermined
127 * destination image.
128 * @param destImg      Image model where result image is to stored.
129 * @param srcImg       Source image model.
130 * @param iters        Number of iterations of the median filter.
131 * @param kSize        Kernel size: dimension of the kernel (i.e., 5 = 5x5, 7 = 7x7,
132 *                     9 = 9x9, etc.).
133 * @param kShape       Kernel shape: element neighbors to include when finding the
134 *                     median.
135 * @param stdDev       Inner-bounds by which to process pixels (pixel values outside
136 *                     this bound will be median filtered).
137 * @param maskFlag    Flag that indicates that the median filtering will be
138 *                     performed for the whole image if equal to true.
139 */
140 public PlugInAlgorithmMedian(ModelImage destImg, ModelImage srcImg, int iters,
141                             int kSize, int kShape, float stdDev, boolean maskFlag) {
142
143     super(destImg, srcImg);
144     if (srcImg.isColorImage() ) {
145         isColorImage = true;
146         valuesPerPixel = 4;
147     }
148     // else, already false
149     entireImage = maskFlag;
150     iterations = iters;
151     kernelSize = kSize; // dimension of the kernel
152     kernelShape = kShape; // set up the mask (kernel) used to filter
153     stdDevLimit = stdDev; // inside magnitude bounds of pixel value to adjust
154     sliceFiltering = true; // as a default--this doesn't make much sense in a 2D
155                           // application.
156     numberOfSlices = 1; // 2D images may only have 1 slice.
157     makeKernel();
158 }
159
160 /**
161 * Constructor for 3D images in which changes are returned to the source image.
162 * @param srcImg       Source image model.
163 * @param iters        Number of iterations of the median filter.
164 * @param kSize        Kernel size: dimension of the kernel (i.e., 5 = 5x5, 7 = 7x7,
165 *                     9 = 9x9, etc.).
166 * @param kShape       Kernel shape: element neighbors to include when finding the
167 *                     median.
168 * @param stdDev       Inner-bounds by which to process pixels (pixel values outside
169 *                     this bound will be median filtered).
170 * @param sliceBySlice Each slice in a volume image is to be filtered separately (when
171 *                     true), else the volume will use a kernel with 3 dimensions.
172 * @param maskFlag    Flag that indicates that the median filtering will be
173 *                     performed for the whole image if equal to true.
174 */
175 public PlugInAlgorithmMedian(ModelImage srcImg, int iters, int kSize, int kShape,
176                             float stdDev, boolean sliceBySlice, boolean maskFlag) {
177     super(null, srcImg);
178     if (srcImg.isColorImage() ) {
179         isColorImage = true;

```

```

180             valuesPerPixel = 4;
181         }
182         // else, already false
183         entireImage = maskFlag;
184         iterations = iters;
185         kernelSize = kSize; // dimension of the kernel
186         kernelShape = kShape; // set up the mask (kernel) used to filter
187         stdDevLimit = stdDev; // inside magnitude bounds of pixel value to adjust
188         sliceFiltering = sliceBySlice;
189         numberOfSlices = srcImage.getExtents()[2];
190         makeKernel();
191     }
192
193 /**
194 * Constructor for 2D images in which changes are returned to the source image.
195 * @param srcImg Source image model.
196 * @param iters Number of iterations of the median filter.
197 * @param kSize Kernel size: dimension of the kernel (i.e., 5 = 5x5, 7 = 7x7,
198 * 9 = 9x9, etc.).
199 * @param kShape Kernel shape: element neighbors to include when finding the
200 * * median.
201 * @param stdDev Inner-bounds by which to process pixels (pixel values outside
202 * this bound will be median filtered).
203 * @param maskFlag Flag that indicates that the median filtering will be
204 * performed for the whole image if equal to true.
205 */
206 public PlugInAlgorithmMedian(ModelImage srcImg, int iters, int kSize, int kShape,
207 float stdDev, boolean maskFlag) {
208     super(null, srcImg);
209     if (srcImg.isColorImage() ) {
210         isColorImage = true;
211         valuesPerPixel = 4;
212     }
213     // else, already false
214     entireImage = maskFlag;
215     iterations = iters;
216     kernelSize = kSize; // dimension of the kernel
217     kernelShape = kShape; // set up the mask (kernel) used to filter
218     stdDevLimit = stdDev; // inside magnitude bounds of pixel value to adjust
219     sliceFiltering = true; // as a default--though a different value doesn't
220     // make much sense in a 2D application.
221     // (calculates makeKernel() )
222     numberOfSlices = 1; // 2D images may only have 1 slice.
223     makeKernel();
224 }
225 /**
226 * RGB images are median filtered by "channel." That is, each color,
227 * red, blue and green, is filtered independently of the other two colors.
228 * This median filter permits selectively filtering any combination of the
229 * three channels instead of simply filtering all three. True for any of
230 * the arguments enforces filtering that channel.
231 * @param rFilter red channel.
232 * @param gFilter green channel.
233 * @param bFilter blue channel.
234 */
235 public void setRGBChannelFilter(boolean r, boolean g, boolean b) {
236     if (isColorImage) { // just in case somebody called for a mono image
237         rChannel = r;
238         gChannel = g;

```

```

239             bChannel = b;
240         }
241     }
242
243     /**
244      * Prepares this class for destruction.
245     */
246     public void finalize(){
247         destImage = null;
248         srcImage = null;
249         super.finalize();
250     }
251
252     /**
253      * Constructs a string of the construction parameters and outputs the string to the
254      * message frame if the logging procedure is turned on.
255     */
256     private void constructLog() {
257         historyString = new String( "Median(" +
258             String.valueOf(kernelShape) + ", " +
259             String.valueOf(kernelSize) + ", " +
260             String.valueOf(iterations) + ", " +
261             String.valueOf(entireImage) + ")\\n");
262     }
263
264     /**
265      * Starts the algorithm.
266     */
267     public void runAlgorithm() {
268
269         if (srcImage == null) {
270             displayError("Source Image is null");
271             return;
272         }
273
274         constructLog();
275
276         if (destImage != null){ // if there exists a destination image
277             if (srcImage.getNDims() == 2){
278                 calcStoreInDest2D();
279             }
280             else if (srcImage.getNDims() > 2) {
281                 calcStoreInDest3D();
282             }
283         }
284         else { // there is no image but the original source.
285             if (srcImage.getNDims() == 2){
286                 calcInPlace2D();
287             }
288             else if (srcImage.getNDims() > 2) {
289                 calcInPlace3D();
290             }
291         }
292     } // end runAlgorithm()
293
294     /**
295      * Median filters the source image. Replaces the original image with the filtered
296      * image.
297     */
298     private void calcInPlace2D(){

```

```

299
300         int length;           // total number of data-elements (pixels) in image
301         float buffer[];       // data-buffer (for pixel data) which is the "heart"
302         of the
303         float resultBuffer[]; // image
304         after               // copy-to buffer (for pixel data) for image-data
305                     // filtering
306
307     try {
308         if (!isColorImage) {
309             // image length is length in 2 dims
310             length = srcImage.getExtents()[0]
311                         *srcImage.getExtents()[1];
312         }
313         else { // if (isColorImage) {
314             // image length is length in 2 dims
315             // by 4 color elements per pixel
316             length = srcImage.getExtents()[0]
317                         *srcImage.getExtents()[1]
318                         *4; // 1 each for ARGB
319         }
320         buffer      = new float[length];
321         resultBuffer = new float[length];
322         srcImage.exportData(0,length, buffer); // locks and releases lock
323     }
324     catch (IOException error) {
325         buffer = null;
326         resultBuffer = null;
327         errorCleanUp("Algorithm Median: source image locked", true);
328         return;
329     }
330     catch (OutOfMemoryError e){
331         buffer = null;
332         resultBuffer = null;
333         errorCleanUp("Algorithm Median reports: Out of memory when creating image
334         buffer", true);
335         return;
336     }
337     this.buildProgressBar();           // let user know what is going on
338     this.sliceFilter(buffer, resultBuffer, 0, "image"); // filter this slice
339     disposeProgressBar();           // filtering work should be done.
340
341     if (threadStopped) {
342         finalize();
343         return;
344     }
345
346     try { // place buffer data into the image
347         srcImage.importData(0, resultBuffer, true);
348     }
349     catch (IOException error) {
350         buffer = null;
351         resultBuffer = null;
352         errorCleanUp("Algorithm Median: Source image locked", true);
353         return;
354     }
355     setCompleted(true);

```

```

357     }
358
359     /**
360      * Median filters the source image and replaces the source image with the median
361      * filtered image.
362     */
363     private void calcInPlace3D(){
364
365         int imageSliceLength = srcImage.getExtents()[0]*srcImage.getExtents()[1];
366         int length;
367         float buffer[];
368         float resultBuffer[];
369
370         try {
371             if (!isColorImage) {
372                 // image length is length in 3 dims
373                 length = srcImage.getExtents()[0]
374                     *srcImage.getExtents()[1]
375                     *srcImage.getExtents()[2];
376             }
377             else { // if (isColorImage) {
378                 // image length is length in 3 dims
379                 // by 4 color elements per pixel
380                 length = srcImage.getExtents()[0]
381                     *srcImage.getExtents()[1]
382                     *srcImage.getExtents()[2]
383                     *4; // 1 each for ARGB
384             }
385             buffer      = new float[length];
386             resultBuffer = new float[length];
387             srcImage.exportData(0,length, buffer); // locks and releases lock
388             this.buildProgressBar();
389         }
390         catch (IOException error) {
391             buffer = null;
392             resultBuffer = null;
393             errorCleanUp("Algorithm Median: Source image locked", true);
394             return;
395         }
396         catch (OutOfMemoryError e){
397             buffer      = null;
398             resultBuffer = null;
399             errorCleanUp("Algorithm Median: Out of memory", true);
400             return;
401         }
402         if (sliceFiltering){
403             for (currentSlice = 0; currentSlice < number_of_slices && !threadStopped;
404                 currentSlice++){
405                 sliceFilter(buffer, resultBuffer, currentSlice*imageSliceLength,
406                             "slice " + String.valueOf(currentSlice+1));
407             }
408         }
409         else { // volume kernel requested
410             volumeFilter(buffer, resultBuffer);
411         }
412
413         if (threadStopped) {
414             finalize();
415             return;

```

```

416         }
417
418     try {
419         srcImage.importData(0, resultBuffer, true);
420     }
421     catch (IOException error) {
422         buffer = null;
423         resultBuffer = null;
424         errorCleanUp("Algorithm Median: source image locked", true);
425         setThreadStopped(true);
426         return;
427     }
428
429     progressBar.dispose();
430     setCompleted(true);
431 }
432
433 /**
434 * This function produces a new image that has been median filtered and places
435 * filtered image in the destination image.
436 */
437 private void calcStoreInDest2D(){
438
439     int length;                                // total number of data-elements (pixels) in image
440     float buffer[];                            // data-buffer (for pixel data) which is the "heart"
441     of                                         // the image
442     float resultBuffer[];                      // copy-to buffer (for pixel data) for image data
443     after                                       // filtering
444
445     try { destImage.setLock(ModelStorageBase.RW_LOCKED); }
446     catch (IOException error){
447         errorCleanUp("Algorithm Median reports: destination image locked", false);
448         return;
449     }
450     try {
451         if (!isColorImage) {
452             // image length is length in 2 dims
453             length = srcImage.getExtents()[0]
454                 *srcImage.getExtents()[1];
455         }
456         else { // if (isColorImage) {
457             // image length is length in 2 dims
458             // by 4 color elements per pixel
459             length = srcImage.getExtents()[0]
460                 *srcImage.getExtents()[1]
461                 *4; // 1 each for ARGB
462         }
463         buffer      = new float[length];
464         resultBuffer = new float[length];
465         srcImage.exportData(0,length, buffer); // locks and releases lock
466     }
467     catch (IOException error) {
468         buffer = null;
469         resultBuffer = null;
470         errorCleanUp("Algorithm Median reports: source image locked", true);
471         return;
472     }
473     catch (OutOfMemoryError e){

```

```

474         buffer = null;
475         resultBuffer = null;
476         errorCleanUp("Algorithm Median reports: out of memory", true);
477         return;
478     }
479
480     this.buildProgressBar();
481     sliceFilter(buffer, resultBuffer, 0, "image"); // filter image based on provided
482     info destImage.releaseLock(); // we didn't want to allow the image to
483     be // be adjusted by someone else
484     progressBar.dispose();
485
486     if (threadStopped) {
487         finalize();
488         return;
489     }
490
491     try { // but now place buffer data into the
492         image destImage.importData(0, resultBuffer, true);
493     }
494     catch (IOException error) {
495         buffer = null;
496         resultBuffer = null;
497         errorCleanUp("Algorithm Median reports: destination image still locked",
498                     true);
499         return;
500     }
501
502     setCompleted(true);
503 }
504
505 /**
506 * This function produces a new volume image that has been median filtered.
507 * Image can be filtered by filtering each slice individually, or by filtering using
508 * a kernel-volume.
509 */
510 private void calcStoreInDest3D(){
511
512     int length;
513     int imageSliceLength = valuesPerPixel *
514         srcImage.getExtents()[0]*srcImage.getExtents()[1]; // cover case of color image
515     float buffer[];
516     float resultBuffer[];
517
518     try { destImage.setLock(ModelStorageBase.RW_LOCKED); }
519     catch (IOException error){
520         errorCleanUp("Algorithm Median reports: destination image locked", false);
521         return;
522     }
523     try {
524         if (!isColorImage) {
525             // image length is length in 3 dims
526             length = srcImage.getExtents()[0]
527                         *srcImage.getExtents()[1]
528                         *srcImage.getExtents()[2];
529         }
530         else { // if (isColorImage) {

```

```

531             // image length is length in 3 dims
532             // by 4 color elements per pixel
533             length = srcImage.getExtents()[0]
534                     *srcImage.getExtents()[1]
535                     *srcImage.getExtents()[2]
536                     *4; // 1 each for ARGB
537         }
538         buffer = new float[length];
539         srcImage.exportData(0,length, buffer); // locks and releases lock
540         this.buildProgressBar();
541     }
542     catch (IOException error) {
543         buffer = null;
544         resultBuffer = null;
545         errorCleanUp("Algorithm Median: source image locked", true);
546         return;
547     }
548     catch (OutOfMemoryError e){
549         buffer = null;
550         resultBuffer = null;
551         errorCleanUp("Algorithm Median: Out of memory creating process buffer",
552                     true);
553         return;
554     }
555
556     try { resultBuffer = new float[length];}
557     catch(OutOfMemoryError e){
558         buffer = null;
559         resultBuffer = null;
560         errorCleanUp("Algorithm Median reports: Out of memory because of
561                     resultBuffer", true);
562         return;
563     }
564
565     if (sliceFiltering){
566         for ( currentSlice = 0; currentSlice < numberoSlices && !threadStopped;
567                 currentSlice++) {
568             sliceFilter(buffer, resultBuffer, currentSlice*imageSliceLength,
569                         "slice "+String.valueOf(currentSlice+1));
570         }
571     }
572     else { // requested volume filter
573         if (isColorImage) // for color image
574             volumeColorFilter(buffer, resultBuffer);
575         else // for mono image
576             volumeFilter(buffer, resultBuffer);
577     }
578
579     destImage.releaseLock();
580
581     if (threadStopped) {
582         finalize();
583         return;
584     }
585
586     try{destImage.importData(0, resultBuffer, true);}
587     catch (IOException e)
588     {
589         buffer = null;
590         resultBuffer = null;

```

---

```

591         errorCleanUp("Algorithm Median reports: destination image still locked",
592                         true);
593         return;
594     }
595
596     progressBar.dispose();
597     setCompleted(true);
598 }
599
600 /**
601 * Allows a single slice to be filtered. Note that a progressBar must be created
602 * first.
603 * @param srcBuffer           Source buffer.
604 * @param destBuffer          Destination Buffer.
605 * @param bufferStartingPoint Starting point for the buffer.
606 * @param msgString           A text message that can be displayed as a message text
607 *                           in the progressBar.
608 */
609 private final void sliceFilter(float srcBuffer[],
610                               float destBuffer[],
611                               int bufferStartingPoint,
612                               String msgString) {
613     int i, a, pass;
614                                         // counting.... i is the offset
615                                         // from the bufferStartingPoint
616     // a adds support for 3D filtering by counting is as the pixel at the starting
617     // point plus the counter offset
618     int buffStart = bufferStartingPoint;           // data element at the buffer. a =
619                                         // bufferStartingPoint+i
620     int sliceLength = srcImage.getSliceSize();
621     int imageSliceLength = sliceLength * valuesPerPixel; // since there are 4 values
622                                         // for every color pixel.
623     int kCenter = maskCenter;                      // to find the middle pixel of the kernel-
mask
624     int width = srcImage.getExtents()[0];           // width of slice in number of pixels (
625     int height = srcImage.getExtents()[1];           // height of slice in number of pixels
626     int sliceWidth = width * valuesPerPixel; // width of slice, which, in color images
is
627                                         // (4*width)
628     int sliceHeight = height;                      // height of image, which, actually
doesn't
629                                         // change
630     int initialIndex = 0;                          // first element is alpha
631
632     float tempBuffer[];
633
634     float average;                            // arithmetic mean
635     float sigma;                             // standard deviation
636
637     float maskedList[];                      // list of buffer-values that were
638                                         // showing inside the mask
639     int row, col;                           // row and column vars for easier
640                                         // reading [(0,0) is in the top-left
641                                         // corner]
642     int mod;                                // 1% length of slice for percent
643                                         // complete
644
645     // these bounds "frame" the interior of the slice which may be filtered
646     // (&adjusted);
647     // image outside the frame may not

```

```

648     int upperBound, lowerBound,      // bounds on the row
649         leftBound, rightBound;    // bounds on the column
650
651     if (isColorImage) {
652         upperBound = halfK;
653         leftBound = halfK*4;
654         lowerBound = sliceHeight - halfK - 1;
655         rightBound = sliceWidth - halfK*4 - 1;
656
657         // data element at the buffer (a = i+bufferStartingPoint) must start on an
658         // alpha value
659         buffStart = bufferStartingPoint - bufferStartingPoint%4; // & no effect if
660                                         // bufferStartingPoint%4 ==
661                                         0 !!!
662
663         // copy all alpha values in this slice
664         setCopyColorText("alpha");
665         for (a = buffStart, i = 0; i < imageSliceLength; a+=4, i+=4) {
666             destBuffer[a] = srcBuffer[a];           // copy alpha;
667         }
668     }
669     else {          // monochrome image
670         upperBound = leftBound = halfK;
671         rightBound = sliceWidth - halfK - 1;
672         lowerBound = sliceHeight - halfK - 1;
673     }
674     mod = (imageSliceLength*numberOfSlices)/100; // mod is 1 percent of length of slice
+
675                                         // the number of slices.
676
677     BitSet mask = srcImage.generateVOIMask();
678
679     for (pass = 0; pass < iterations && !threadStopped; pass++) {
680         a = buffStart;                      // set/reset a to address pixels
681                                         // from the beginning of this
682                                         // buffer.
683         if (isColorImage) {                // color image dealt with in
684                                         // special way
685                                         // choose i so the proper colors go
686                                         // copy only needed RGB values
687                                         // start with alpha on each pass
688                                         // (routine moved so we don't do
689                                         // it for each pass)
690         while (initialIndex < 3 && !threadStopped) { // alpha:0, R:1, G:2,
691                                         // B:3. But alpha must
692                                         // be copied
693             ++initialIndex;                  // next initial index
694             a += initialIndex;              // keep the pixel location up with
695                                         // color indexed to
696
697             if (numberOfSlices > 1 && pBarVisible == true) { // 3D image update
698                                         // progressBar
699                                         // do a progress bar update
700             progressBar.setValue(Math.round
701                                         (((float)(3*currentSlice*iterations + 3*pass +
702                                         (initialIndex - 1))/(3*iterations*numberOfSlices))*100));
703         }
704     }
705

```

```

706             if (!rChannel && initialIndex==1) {
707                 // when looking at the image reds but we're not filtering the red
708                 channel
709                     // copy all red values
710                     setCopyColorText("red");
711                     for (i = initialIndex; i < imageSliceLength; a+= 4, i+=4) {
712                         destBuffer[a] = srcBuffer[a];
713                     }
714             } else if (!gChannel && initialIndex==2) {
715                 // when looking at the image greens but we're not filtering the
716                 // greens channel
717                     // copy all greens values
718                     setCopyColorText("green");
719                     for (i = initialIndex; i < imageSliceLength; a+=4, i+=4) {
720                         destBuffer[a] = srcBuffer[a];
721                     }
722             } else if (!bChannel && initialIndex==3) {
723                 // when looking at the image blues but we're not filtering the
724                 // blues channel
725                     // copy all blue values
726                     setCopyColorText("blue");
727                     for (i = initialIndex; i < imageSliceLength; a+=4, i+=4) {
728                         destBuffer[a] = srcBuffer[a];
729                     }
730             }
731         } else {
732             if (pBarVisible == true) {
733                 progressBar.setMessage("Filtering " + msgString + " (pass " +
734                     String.valueOf(pass+1) + " of " + iterations +") ...");
735             }
736             // if we needed to filter the image, we dropped through the
737             // selection to filter the
738                 // color given by ints initialIndex
739                     for (i = initialIndex; i < imageSliceLength && !threadStopped;
740                         a+=4, i+=4){
741                         if (numberOfSlices == 1) { // 2D image update progressBar
742                             if (i%mod == 0 && pBarVisible == true) {
743                                 progressBar.setValue(Math.round
744                                     ( (float)(3*(pass*sliceLength) + (initialIndex-
745                                         1)*sliceLength + i/4)/
746                                         (3*iterations*(sliceLength-1))*100) );
747                             }
748                         }
749                     }
750                     if (entireImage == true || mask.get(a/4) ) { // may have problems
751                         // in masking ...
752                         row = i/sliceWidth;
753                         col = i%sliceWidth;
754                         if ( (row < upperBound) || (row > lowerBound) ) {
755                             destBuffer[a] = srcBuffer[a]; // row too far up or
756                                         // down--out of bounds
757                         }
758                         else if ((col < leftBound) || (col > rightBound)) {
759                             destBuffer[a] = srcBuffer[a]; // column too far left
760                                         // or right--out of bounds
761                         }
762                         else { // in bounds
763                             maskedList = getNeighborList(a, srcBuffer, true);
764                             // verify that this element is an outlier

```

```

765                         if (stdDevLimit == 0.0) { // anything is an outlier
766                             shellSort(maskedList);
767                             destBuffer[a] = median(maskedList);
768                         }
769                         else { // look for outlierness
770                             average = mean(maskedList);
771                             sigma = standardDeviation(maskedList, average);
772                             if ((maskedList[kCenter] > (average +
773                                 stdDevLimit*sigma)) ||
774                                 (maskedList[kCenter] < (average -
775                                 stdDevLimit*sigma))) {
776                                 shellSort(maskedList);
777                                 destBuffer[a] = median(maskedList);
778                             }
779                         else { // if element was not an outlier, pixel is
780                             destBuffer[a] = srcBuffer[a];
781                         }
782                     }
783                 }
784             }
785         destination
786             // buffer.
787             destBuffer[a] = srcBuffer[a];
788         }
789     }
790     a = buffStart;
791         // reset the index back to the beginning of
792         // the filterarea
793     }
794 }
795 else { // monochrome image
796     if (pBarVisible) {
797         progressBar.setMessage("Filtering " + msgString + " (pass " +
798             String.valueOf(pass+1) + " of " + iterations +") ...");
799         if (numberOfSlices > 1) { // 3D image update progressBar
800             // do a progress bar update
801             progressBar.setValue(Math.round
802                 ((( (float)(currentSlice*iterations + pass)/
803                     (iterations*numberOfSlices))* 100)));
804         }
805     }
806     for ( i = 0; i < imageSliceLength && !threadStopped; i++){
807         if (numberOfSlices == 1) { // 2D image update progressBar
808             if (i%mod == 0 && pBarVisible == true) {
809                 progressBar.setValue(Math.round
810                     ((( (float)((pass*imageSliceLength)+i)/(iterations*(imageSliceLength
811                         -1))*100)) );
812             }
813         }
814         if (entireImage == true || mask.get(a) ) {
815             // Median stuff here
816             row = i/width;
817             col = i%width;
818             if ( (row < upperBound) || (row > lowerBound) ) {
819                 destBuffer[a] = srcBuffer[a]; // row too far up or down--out of
820             }
821             else if ((col < leftBound) || (col > rightBound)) {

```

```

822                     destBuffer[a] = srcBuffer[a]; // column too far left or right--
823                     out of
824                     bounds
825                 }
826             else { // in bounds
827                 maskedList = getNeighborList(a, srcBuffer, true);
828                 // verify that this element is an outlier
829                 if (stdDevLimit == 0.0) { // anything is an outlier
830                     shellSort(maskedList);
831                     destBuffer[a] = median(maskedList);
832                 }
833                 else { // look for outlierness
834                     average = mean(maskedList);
835                     sigma = standardDeviation(maskedList, average);
836                     if ((maskedList[kCenter] > (average + stdDevLimit*sigma)) ||
837                         (maskedList[kCenter] < (average - stdDevLimit*sigma))) {
838                         shellSort(maskedList);
839                         destBuffer[a] = median(maskedList);
840                     }
841                     else { // if element was not an outlier, pixel is fine.
842                         destBuffer[a] = srcBuffer[a];
843                     }
844                 }
845             }
846         else { // not part of the VOI so just copy this into the
847             destination
848             buffer.
849             destBuffer[a] = srcBuffer[a];
850             a++; // address the next data element from the
851             bufferStartingPoint
852         }
853     }
854     // now set up for the repeat for multiple iterations.
855     // But only bother with copying over if there are more iterations.
856     if (pass < iterations - 1) {
857         tempBuffer = destBuffer; // swap dest & src buffers
858         destBuffer = srcBuffer;
859         srcBuffer = tempBuffer;
860     }
861 }
862 // destBuffer should now be copied over for the size of imageSliceLength. You
863 may
864 // return.
865 }
866 /**
867 * Filter a 3D image with a 3D kernel. Allows median filtering to include the
868 picture
869 * elements at greater depths than only the current slice.
870 * <p><em>Note that this volume filter will correctly filter color images on all
871 bands (aRGB)
872 * because the neighbor list is correct (see getNeighborList()). This means,
however, it
873 * will not selectively filter any bands (one may not filter only the Red channel,
for
874 * instance), and will also filter all alpha values as well. Of course, progress
bar updates

```

```

873     * will not include any color information.
874     * For these reasons it a useable, but limited color filter.</em>
875     * @param srcBufferSource image.
876     * @param destBufferDestination image.
877     * @see volumeColorFilter
878     * @see getNeighborList
879     *
880     */
881     // some code has been left in to allow this method to properly filter
882     // color images, although the other method is included.
883     private void volumeFilter(float srcBuffer[], float destBuffer[]) {
884         int i, pass;           // counting the current element
885         int row,              // ease of reading to find the row, column and slice
886             column,            // (all starting at 0) associated with the current element
887             slice;              // [(0,0,0) starts at the closest upper-left corner]
888         int imageSliceLength = srcImage.getSliceSize() * valuesPerPixel;
889         int imageLength = imageSliceLength * numberoSlices;
890         int kCenter = maskCenter;
891         int width = srcImage.getExtents()[0];      // width of slice in number of pixels
892         int height = srcImage.getExtents()[1];       // height of slice in number of pixels
893         int sliceWidth = width*valuesPerPixel;        // width of slice in number of intensity
894                                         // values (as in colors per pixel (1 for
895         mono,                                // 4 for color))
896         float tempBuffer[];
897
898         float average;          // arithmetic mean
899         float sigma;            // standard deviation
900
901         float maskedList[];      // list of buffer-values that were showing inside the
902         mask
903
904         // these bounds "frame" the interior of the slice which may be filtered
905         // (&adjusted);
906         // image outside the frame may not
907         int leftBound, rightBound,           // bounds on the column
908             upperBound, lowerBound,          // bounds on the row
909             aheadBound, behindBound;        // bounds on the slice
910
911         // (a note on orientation: object front is facing in the same direction as
912         // viewer, thus ahead of viewer is into monitor, behind is out of monitor and
913         // a more positive number of slices is farther forward.)
914         upperBound = halfK;
915         lowerBound = height - halfK -1;
916         behindBound = halfK;
917         aheadBound = numberoSlices - halfK - 1;
918
919         // we may say that each column is a pixel intensity: mono images have 1 per
920         // pixel, 4 in
921         // color;
922         // these calculations are done separately for color & mono images in
923         sliceFilter().
924
925         leftBound = halfK * valuesPerPixel;
926         rightBound = sliceWidth - valuesPerPixel*halfK - 1; // in color: (4*width - 4*halfK
- 1);                                              //mono: (width - halfK - 1)
927
928         int mod = (imageLength)/100; // mod is 1 percent of length of slice * the number of
929                                         // slices.
930
931         BitSet mask = srcImage.generateVOIMask();
932
933

```

```

927         for (pass = 0; pass < iterations && !threadStopped; pass++) {
928             if (pBarVisible == true) {
929                 progressBar.setMessage("Filtering image (pass "+ String.valueOf(pass+1) +" of "+ iterations +") ...");
930             }
931             for ( i = 0; i < imageLength && !threadStopped; i++){
932                 if (i%mod == 0 && pBarVisible == true) {
933                     progressBar.setValue(Math.round
934                         ( ( float)( (pass*imageLength)+i)/(iterations*(imageLength-1))*100) );
935                 }
936
937                 if (entireImage == true || mask.get(i/valuesPerPixel) ) {
938                     // Median stuff here
939                     slice = i/imageSliceLength;
940                     row = (i%imageSliceLength)/sliceWidth;
941                     column = i%sliceWidth;
942
943                     if ( (row < upperBound) || (row > lowerBound) ) {
944                         destBuffer[i] = srcBuffer[i]; // row too far up or down--out of bounds
945                     }
946                     else if ((column < leftBound) || (column > rightBound)) {
947                         destBuffer[i] = srcBuffer[i]; // column too far left or right--out of
948                                         // bounds
949                     }
950                     else if ((slice < behindBound) || (slice > aheadBound)) {
951                         destBuffer[i] = srcBuffer[i]; // slice too far ahead or behind--out of
952                                         // bounds
953                     }
954                     else { // in bounds
955                         maskedList = getNeighborList(i, srcBuffer, false);
956                         // verify that this element is an outlier
957                         if (stdDevLimit == 0.0) { // anything is an outlier
958                             shellSort(maskedList);
959                             destBuffer[i] = median(maskedList);
960                         }
961                         else { // look for outlierness
962                             average = mean(maskedList);
963                             sigma = standardDeviation(maskedList, average);
964                             if (((maskedList[kCenter] > (average + stdDevLimit*sigma)) ||
965                                 (maskedList[kCenter] < (average - stdDevLimit*sigma))) {
966                                 shellSort(maskedList);
967                                 destBuffer[i] = median(maskedList);
968                             }
969                             else { // if element was not an outlier, pixel is fine.
970                                 destBuffer[i] = srcBuffer[i];
971                             }
972                         }
973                     }
974                 }
975                 else { // not part of the VOI so just copy this into the destination
976                     destBuffer[i] = srcBuffer[i];
977                 }
978             }
979             // now set up for the repeat for multiple iterations.
980             // But only bother with copying over if there are more iterations.
981             if (pass < iterations - 1) {
982                 tempBuffer = destBuffer; // swap src & dest buffer
983                 destBuffer = srcBuffer;
984                 srcBuffer = tempBuffer;

```

```

985         }
986     }
987 }
988 /**
989 * Filter a Color 3D image with a 3D kernel. Allows median filtering to
990 * include the picture elements at greater depths than only the current
991 * slice. This method allows selected band filtering, and does not filter
992 * the alpha band.
993 * @param srcBufferSource image.
994 * @param destBufferDestination image.
995 * @see volumeFilter
996 *
997 */
998
999 private void volumeColorFilter(float srcBuffer[], float destBuffer[]) {
1000     int i, pass;           // counting the current element
1001     int initialIndex;    // reference to the color band being filtered/copied: aRGB: 0,
1, 2,
1002             3;
1003             // it is an offset to the identified pixel, or column, of the
slice
1004     int row,               // ease of reading to find the row, column and slice
1005     column,              // (all starting at 0) associated with the current element
1006     slice;                // [(0,0,0) starts at the closest upper-left corner]
1007     int kCenter = maskCenter;
1008     int width = srcImage.getExtents()[0];      // width of slice in number of pixels
1009     int height = srcImage.getExtents()[1];       // height of slice in number of pixels
1010     int sliceWidth = width*valuesPerPixel;        // width of slice in number of intensity
1011                                         // values
1012                                         // (as in colors per pixel (1 for mono, 4 for
1013                                         // color))
1014     int sliceSize = width * height;              // in pixels (or elements)
1015     int imageSliceLength = width * height * valuesPerPixel; // in values-pixels
1016     int imageSize = sliceSize * numberofslices; // in pixels (or elements)
1017     int imageLength = imageSliceLength * numberofslices; // in (values-pixels)
1018     float tempBuffer[];
1019
1020     float average;            // arithmetic mean
1021     float sigma;             // standard deviation
1022
1023     float maskedList[];      // list of buffer-values that were showing inside the
mask
1024
1025     // these bounds "frame" the interior of the slice which may be filtered
(&adjusted);
1026     // image outside the frame may not
1027     int leftBound, rightBound,          // bounds on the column
1028         upperBound, lowerBound,        // bounds on the row
1029         aheadBound, behindBound;      // bounds on the slice
1030     // (a note on orientation: object front is facing in the same direction as
1031     // viewer, thus ahead of viewer is into monitor, behind is out of monitor and
1032     // a more positive number of slices is farther forward.)
1033     upperBound = halfK;
1034     lowerBound = height - halfK - 1;
1035     behindBound = halfK;
1036     aheadBound = numberofslices - halfK - 1;
1037     // we may say that each column is a pixel intensity: mono images have 1 per
pixel, 4 in
1038     // color;

```

```

1039      // these calculations are done separately for color & mono images in
1040      sliceFilter().
1041      leftBound = halfK * valuesPerPixel;
1042      rightBound = sliceWidth - valuesPerPixel*halfK - 1; // in color: (4*width - 4*halfK
1043      - 1);
1044      // mono: (width - halfK - 1)
1045
1046      int mod = (imageSize)/100; // mod is 1 percent of length of slice * the number of
1047      slices.
1048
1049      BitSet mask = srcImage.generateVOIMask();
1050
1051      // copy all alpha values in the image
1052      setCopyColorText("alpha");
1053      for (i = 0; i < imageLength; i+=4) {
1054          destBuffer[i] = srcBuffer[i]; // copy alpha;
1055      }
1056
1057      // choose i so the proper colors go alongside the initial index
1058      // so we get the right output statements in the progress bar
1059      // copy only needed RGB values
1060      initialIndex = 0; // start with alpha on each pass (routine moved so
1061      // we don't
1062      // do it for each pass)
1063      while (initialIndex < 3 && !threadStopped) { // alpha:0, R:1, G:2, B:3. But
1064          alpha // must be copied
1065          ++initialIndex; // next initial index
1066
1067          if (!rChannel && initialIndex==1) {
1068              // when looking at the image reds but we're not filtering the red channel
1069              // copy all red values
1070              setCopyColorText("red");
1071              for (i = initialIndex; i < imageLength; i+=4) {
1072                  destBuffer[i] = srcBuffer[i];
1073              }
1074          }
1075          else if (!gChannel && initialIndex==2) {
1076              // when looking at the image greens but we're not filtering the greens
1077              // copy all greens values
1078              setCopyColorText("green");
1079              for (i = initialIndex; i < imageLength; i+=4) {
1080                  destBuffer[i] = srcBuffer[i];
1081              }
1082          }
1083          else if (!bChannel && initialIndex==3) {
1084              // when looking at the image blues but we're not filtering the blues
1085              // copy all blue values
1086              setCopyColorText("blue");
1087              for (i = initialIndex; i < imageLength; i+=4) {
1088                  destBuffer[i] = srcBuffer[i];
1089              }
1090          }
1091      }
1092      else {
1093          for (pass = 0; pass < iterations && !threadStopped; pass++) {
1094              if (pBarVisible == true) {
1095                  if (initialIndex == 1) {
1096                      progressBar.setMessage("Filtering red channel (pass " +
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
20100
20101
20102
20103
20104
20105
20106
20107
20108
20109
20110
20111
20112
20113
20114
20115
20116
20117
20118
20119
20120
20121
20122
20123
20124
20125
20126
20127
20128
20129
20130
20131
20132
20133
20134
20135
20136
20137
20138
20139
20140
20141
20142
20143
20144
20145
20146
20147
20148
20149
20150
20151
20152
20153
20154
20155
20156
20157
20158
20159
20160
20161
20162
20163
20164
20165
20166
20167
20168
20169
20170
20171
20172
20173
20174
20175
20176
20177
20178
20179
20180
20181
20182
20183
20184
20185
20186
20187
20188
20189
20190
20191
20192
20193
20194
20195
20196
20197
20198
20199
20200
20201
20202
20203
20204
20205
20206
20207
20208
20209
20210
20211
20212
20213
20214
20215
20216
20217
20218
20219
20220
20221
20222
20223
20224
20225
20226
20227
20228
20229
20230
20231
20232
20233
20234
20235
20236
20237
20238
20239
20240
20241
20242
20243
20244
20245
20246
20247
20248
20249
20250
20251
20252
20253
20254
20255
20256
20257
20258
20259
202510
202511
202512
202513
202514
202515
202516
202517
202518
202519
202520
202521
202522
202523
202524
202525
202526
202527
202528
202529
202530
202531
202532
202533
202534
202535
202536
202537
202538
202539
202540
202541
202542
202543
202544
202545
202546
202547
202548
202549
202550
202551
202552
202553
202554
202555
202556
202557
202558
202559
202560
202561
202562
202563
202564
202565
202566
202567
202568
202569
202570
202571
202572
202573
202574
202575
202576
202577
202578
202579
202580
202581
202582
202583
202584
202585
202586
202587
202588
202589
202590
202591
202592
202593
202594
202595
202596
202597
202598
202599
2025100
2025101
2025102
2025103
2025104
2025105
2025106
2025107
2025108
2025109
2025110
2025111
2025112
2025113
2025114
2025115
2025116
2025117
2025118
2025119
2025120
2025121
2025122
2025123
2025124
2025125
2025126
2025127
2025128
2025129
2025130
2025131
2025132
2025133
2025134
2025135
2025136
2025137
2025138
2025139
2025140
2025141
2025142
2025143
2025144
2025145
2025146
2025147
2025148
2025149
2025150
2025151
2025152
2025153
2025154
2025155
2025156
2025157
2025158
2025159
20251510
20251511
20251512
20251513
20251514
20251515
20251516
20251517
20251518
20251519
20251520
20251521
20251522
20251523
20251524
20251525
20251526
20251527
20251528
20251529
20251530
20251531
20251532
20251533
20251534
20251535
20251536
20251537
20251538
20251539
20251540
20251541
20251542
20251543
20251544
20251545
20251546
20251547
20251548
20251549
20251550
20251551
20251552
20251553
20251554
20251555
20251556
20251557
20251558
20251559
20251560
20251561
20251562
20251563
20251564
20251565
20251566
20251567
20251568
20251569
20251570
20251571
20251572
20251573
20251574
20251575
20251576
20251577
20251578
20251579
20251580
20251581
20251582
20251583
20251584
20251585
20251586
20251587
20251588
20251589
20251590
20251591
20251592
20251593
20251594
20251595
20251596
20251597
20251598
20251599
202515100
202515101
202515102
202515103
202515104
202515105
202515106
202515107
202515108
202515109
202515110
202515111
202515112
202515113
202515114
202515115
202515116
202515117
202515118
202515119
202515120
202515121
202515122
202515123
202515124
202515125
202515126
202515127
202515128
202515129
202515130
202515131
202515132
202515133
202515134
202515135
202515136
202515137
202515138
202515139
202515140
202515141
202515142
202515143
202515144
202515145
202515146
202515147
202515148
202515149
202515150
202515151
202515152
202515153
202515154
202515155
202515156
202515157
202515158
202515159
202515160
202515161
202515162
202515163
202515164
202515165
202515166
202515167
202515168
202515169
202515170
202515171
202515172
202515173
202515174
202515175
202515176
202515177
202515178
202515179
202515180
202515181
202515182
202515183
202515184
202515185
202515186
202515187
202515188
202515189
202515190
202515191
202515192
202515193
202515194
202515195
202515196
202515197
202515198
202515199
202515200
202515201
202515202
202515203
202515204
202515205
202515206
202515207
202515208
202515209
202515210
202515211
202515212
202515213
202515214
202515215
202515216
202515217
202515218
202515219
202515220
202515221
202515222
202515223
202515224
202515225
202515226
202515227
202515228
202515229
202515230
202515231
202515232
202515233
202515234
202515235
202515236
202515237
202515238
202515239
202515240
202515241
202515242
202515243
202515244
202515245
202515246
202515247
202515248
202515249
202515250
202515251
202515252
202515253
202515254
202515255
202515256
202515257
202515258
202515259
202515260
202515261
202515262
202515263
202515264
202515265
202515266
202515267
202515268
202515269
202515270
202515271
202515272
202515273
202515274
202515275
202515276
202515277
202515278
202515279
202515280
202515281
202515282
202515283
202515284
202515285
202515286
202515287
202515288
202515289
202515290
202515291
202515292
202515293
202515294
202515295
202515296
202515297
202515298
202515299
202515300
202515301
202515302
202515303
202515304
202515305
202515306
202515307
202515308
202515309
202515310
202515311
202515312
202515313
202515314
202515315
202515316
202515317
202515318
202515319
202515320
202515321
202515322
202515323
202515324
202515325
202515326
202515327
202515328
202515329
202515330
202515331
202515332
202515333
202515334
202515335
202515336
202515337
202515338
202515339
202515340
202515341
202515342
202515343
202515344
202515345
202515346
202515347
202515348
202515349
20251535
```

```

1092                               String.valueOf(pass+1) + " of "+ iterations +"") ... );
1093
1094             }
1095             else if (initialIndex == 2) {
1096                 progressBar.setMessage("Filtering green channel (pass "+
1097                                         String.valueOf(pass+1) + " of "+ iterations +"") ... );
1098             }
1099             else if (initialIndex == 3) {
1100                 progressBar.setMessage("Filtering blue channel (pass "+
1101                                         String.valueOf(pass+1) + " of "+ iterations +"") ... );
1102             }
1103         }
1104         // if we needed to filter the image, we dropped through the selection
1105         to filter
1106         // the color given by ints initialIndex
1107         for ( i = initialIndex; i < imageLength && !threadStopped; i+=4){
1108             if (i%mod == 0 && pBarVisible == true) {
1109                 progressBar.setValue(Math.round
1110                     ( ( float)(iterations*(initialIndex - 1)*imageSize + imageSize*pass
1111                         + i/4)/(3*iterations*(imageSize-1))*100) );
1112
1113             if (entireImage == true || mask.get(i/valuesPerPixel) ) {
1114                 // Median stuff here
1115                 slice = i/imageSliceLength;
1116                 row = (i%imageSliceLength)/sliceWidth;
1117                 column = i%sliceWidth;
1118
1119                 if ( (row < upperBound) || (row > lowerBound) ) {
1120                     destBuffer[i] = srcBuffer[i]; // row too far up or down--out of
1121                                         // bounds
1122                 }
1123                 else if ((column < leftBound) || (column > rightBound)) {
1124                     destBuffer[i] = srcBuffer[i]; // column too far left or right--
1125                                         // out of bounds
1126                 }
1127                 else if ((slice < behindBound) || (slice > aheadBound)) {
1128                     destBuffer[i] = srcBuffer[i]; // slice too far ahead or behind--
1129                                         // out of bounds
1130                 }
1131                 else { // in bounds
1132                     maskedList = getNeighborList(i, srcBuffer, false);
1133                     // verify that this element is an outlier
1134                     if (stdDevLimit == 0.0) { // anything is an outlier
1135                         shellSort(maskedList);
1136                         destBuffer[i] = median(maskedList);
1137                     }
1138                     else { // look for outlierness
1139                         average = mean(maskedList);
1140                         sigma = standardDeviation(maskedList, average);
1141                         if ((maskedList[kCenter] > (average + stdDevLimit*sigma)) ||
1142                             (maskedList[kCenter] < (average - stdDevLimit*sigma))) {
1143                             shellSort(maskedList);
1144                             destBuffer[i] = median(maskedList);
1145                         }
1146                         else { // if element was not an outlier, pixel is fine.
1147                             destBuffer[i] = srcBuffer[i];
1148                         }
1149                     }
1150                 }
1151             }
1152         }
1153     }
1154 }
```

```

1151           else {      // not part of the VOI so just copy this into the
1152             destination
1153               // buffer.
1154             destBuffer[i] = srcBuffer[i];
1155           }
1156         }
1157       // now set up for the repeat for multiple iterations.
1158       // But only bother with copying over if there are more iterations.
1159     if (pass < iterations - 1) {
1160       tempBuffer = destBuffer;    // swap src & dest buffer
1161       destBuffer = srcBuffer;
1162       srcBuffer = tempBuffer;
1163     }
1164   if (iterations%2 == 0) {      // if even number of iterations, then
1165     tempBuffer = destBuffer;    // swap src & dest buffer is necessary
1166     destBuffer = srcBuffer;    // to keep other colors not-yet-filtered from
1167     srcBuffer = tempBuffer;    // filtering from the wrong buffer,
1168   overwriting
1169   }
1170 }
1171 }
1172 }
1173 /**
1174 * Forms kernel. Note that the <b>kernel</b> uses the 0th place, unlike the
1175 * kernelMask where counting starts at 1.
1176 *
1177 */
1178 private void makeKernel(){
1179   try {
1180     if (sliceFiltering)
1181       kernel = new byte[kernelSize*kernelSize];
1182     else if (!sliceFiltering)
1183       kernel = new byte[kernelSize*kernelSize*kernelSize];
1184   }
1185   catch (OutOfMemoryError e) {
1186     displayError("Algorithm Median reports: not enough memory to form a kernel mask.");
1187     setCompleted(false);
1188     setThreadStopped(true);
1189     return;
1190   }
1191   setKernel();
1192   makeKernelMask();
1193 }
1194 }

1195 /**
1196 * Fill in the mask for which pixels are used in filtering.
1197 *
1198 */
1199 private void setKernel() {
1200   int i;
1201   int halfK = kernelSize/2;
1202
1203   // square/box
1204   if ( (kernelShape == SQUARE_KERNEL) ||
1205       (kernelShape == CUBE_KERNEL)) {
1206     for (i=0; i < kernel.length; i++)
1207
1208

```

```

1209             kernel[i] = 1;
1210         } // end square/cube kernel
1211
1212         // cross/axial
1213     else if ((kernelShape == CROSS_KERNEL) ||
1214               (kernelShape == AXIAL_KERNEL)) {
1215         int row;    // indicates current row
1216         int col;    // indicates current column
1217         if (sliceFiltering) {
1218             for (i = 0; i < kernel.length; i++) {
1219                 row = i/kernelSize;
1220                 col = i%kernelSize;
1221
1222                 if (col == halfK)      {kernel[i] = 1;}
1223                 else if (row == halfK) {kernel[i] = 1;} // should be for a cross -> else
1224             if
1225                 else
1226             }
1227         }
1228     else {      // volume filtering
1229         int slice;
1230         for (i = 0; i < kernel.length; i++) {
1231             slice = i/(kernelSize*kernelSize);
1232             row = (i%(kernelSize*kernelSize))/kernelSize;
1233             col = i%kernelSize;
1234
1235             if (slice == halfK) {
1236                 if (col == halfK)      {kernel[i] = 1;}
1237                 else if (row == halfK) {kernel[i] = 1;}
1238                 else
1239             }
1240             else if ((row == halfK) && (col == halfK)) {
1241                 kernel[i] = 1;
1242             }
1243             else {kernel[i] = 0;}
1244         }
1245     }
1246 } // end cross/axial
1247 else if ( kernelShape == VERT_KERNEL) {
1248     int row;    // indicates current row
1249     int col;    // indicates current column
1250     if (sliceFiltering) {
1251         for (i = 0; i < kernel.length; i++) {
1252             row = i/kernelSize;
1253             col = i%kernelSize;
1254
1255             if (col == halfK)      {kernel[i] = 1;}
1256             else if (row == halfK) {kernel[i] = 0;}
1257             else
1258         }
1259     }
1260     else {      // volume filtering
1261     }
1262 } // end vert
1263 else if ( kernelShape == HORIZ_KERNEL) {
1264     int row;   // indicates current row
1265     int col;   // indicates current column
1266     if (sliceFiltering) {
1267         for (i = 0; i < kernel.length; i++) {

```

```

1268             row = i/kernelSize;
1269             col = i%kernelSize;
1270
1271             if (col == halfK)      {kernel[i] = 0;}
1272             else if (row == halfK) {kernel[i] = 1;}
1273             else                  {kernel[i] = 0;}
1274         }
1275     }
1276     else {      // volume filtering
1277     }
1278 } // end vert
1279
1280 // 'x' kernel
1281 else if (kernelShape == X_KERNEL) {
1282     int row;    // indicates current row
1283     int col;    // indicates current column
1284     int revcol; // runs opposite of the col.
1285     if (sliceFiltering) {
1286         for (i = 0; i < kernel.length; i++) {
1287             row = i/kernelSize;
1288             col = i%kernelSize;
1289             revcol = kernelSize - 1 - col;
1290
1291             if      (row == col)      {kernel[i] = 1;}
1292             else if (row == revcol)  {kernel[i] = 1;}
1293             else                  {kernel[i] = 0;}
1294         }
1295     }
1296     else {      // volume filtering
1297         int slice;
1298         for (i = 0; i < kernel.length; i++) {
1299             slice = i/(kernelSize*kernelSize);
1300             row = (i%(kernelSize*kernelSize))/kernelSize;
1301             col = i%kernelSize;
1302             revcol = kernelSize - 1 - col;
1303
1304             if ((slice == col) ||
1305                 (slice == revcol)) {
1306                 if      (row == col)      {kernel[i] = 1;}
1307                 else if (row == revcol) {kernel[i] = 1;}
1308                 else                  {kernel[i] = 0;}
1309             }
1310             else {kernel[i] = 0;}
1311         }
1312     }
1313 } // end 'x' kernel
1314
1315 }
1316
1317 /**
1318 * Makes the kernel mask.
1319 * The kernel mask is the list of values pulled from the
1320 * image which will be used to find the median of the
1321 * central pixel. Its length is
1322 * <i>(number of pixels to be used to determine median) + 1</i>.
1323 * <p>
1324 * Thus the kernel center (decided here), has the value of
1325 * the location of the central pixel shown in the window.
1326 * The value of the kernel center is the number of pixels picked
1327 * up to median sort.

```

```

1328     *
1329     * Since the kernel mask is <i>number of pixels + 1</i>,
1330     * the maskCenter must be
1331     */
1332     private void makeKernelMask() {
1333         halfK = kernelSize/2;
1334         // figure how many kernel elements are actually in the kernel-mask
1335         int count = 1;// start counting from one, since sort starts with element 1 (even
empty
1336             // mask must have 1 element!)
1337         for (int m = 0; m < kernel.length; m++) {
1338             if (kernel[m] != 0) // if this element is marked 'on'
1339                 count++;
1340         }
1341         kernelMask = new float[count]; // must have the leading element empty: the sort
1342                                     // starts with element 1
1343         if (sliceFiltering) { // 2D
1344
1345             if (kernelShape == SQUARE_KERNEL){
1346                 kernelCenter = count/2 - 1; // whole square
1347                 maskCenter = halfK*(kernelSize + 1) + 1;           // count/2 : I feel dumb
1348             }
1349             else if (kernelShape == CROSS_KERNEL || kernelShape == VERT_KERNEL || kernelShape ==
1350                     HORZ_KERNEL) {
1351                 kernelCenter = halfK*(kernelSize + 1);
1352                 maskCenter = kernelSize;
1353             }
1354             else if (kernelShape == X_KERNEL) { // sizeof kernel is same as CROSS_KERNEL
1355                 kernelCenter = halfK*(kernelSize + 1); // whole square -- (count/2-1)???
1356                 maskCenter = kernelSize;
1357             }
1358         }
1359         else { //3D
1360             if (kernelShape == CUBE_KERNEL) {
1361                 kernelCenter = count/2 - 1;
1362                 maskCenter = halfK*(kernelSize*kernelSize + kernelSize + 1) + 1;
1363             }
1364             else if (kernelShape == AXIAL_KERNEL) {
1365                 kernelCenter = (kernelSize*kernelSize*kernelSize)/2; // whole cube
1366                 maskCenter = count/2; // i feel dumb...
1367             }
1368             else if (kernelShape == X_KERNEL) { // sizeof kernel is same as AXIAL_KERNEL
1369                 kernelCenter = (kernelSize*kernelSize*kernelSize)/2; // whole cube
1370                 maskCenter = count/2; // i feel dumb...
1371             }
1372         }
1373         // not entirely dumb. mc = count/2 because of the symmetry of the mask. custom
masks may
1374         // be diff. & i'd like to include custom masks someday....
1375     }
1376
1377     /**
1378      * Compiles a list of the values neighboring the desired pixel, that are defined
in the
1379      * kernel. Be careful because although the kernel starts its index at 0, the list
that is
1380      * returned starts indexing at 1.
1381      * <p>
1382      * Color images are processed differently from the monochrome images because
although color

```

```

1383     * images use the same size kernel as mono images, it fills the kernel with
1384     * brightness levels
1385     * that are spread out in the data set. The Neighbor list still reports the
1386     * monochromatic
1387     * brightness values. That is, for a color image: the neighbors of the central
1388     * pixel with the
1389     * same color are returned in the neighbor list's kernel.
1390     * @param i The central pixel to find neighbors for.
1391     * @param data Image data
1392     * @param is2D True indicates that the neighbors are found along a
1393                 2D slice (or 2D image) instead of neighbors in a 3D volume.
1394     * @return The neighboring pixel list, where the list starts at 1 (leaving the
1395     * initial
1396     * element 0), and corresponds to the kernel chosen.
1397     */
1398    private final float[] getNeighborList(int i, float[] data, boolean is2D) {
1399        int row, col;
1400        int kCenter = kernelCenter; // index to the central element of the kernel
1401                                         // (this is the mask for which elements in data are
1402                                         used.)
1403        int width = 0; // width of slice in number of pixels
1404        int height = 0; // height of slice in number of pixels
1405
1406        try {
1407            width = srcImage.getExtents()[0];
1408            height = srcImage.getExtents()[1];
1409        } catch (NullPointerException npe) {
1410            Preferences.debug("AlgorithmMedian: null pointer while making neighbor list.");
1411            setThreadStopped(true);
1412            setCompleted(false);
1413        }
1414        int sliceWidth = width * valuesPerPixel; // width of slice in number
1415        of // elements
1416
1417        // place all the masked 'on' elements into the data-list
1418        int count = 1;
1419        // color images are different from the mono images in that though color images
1420        // use the
1421        // same size kernel as mono images, but fill it with brightness levels that are
1422        // spread out
1423        // in the data set.
1424        if (isColorImage) { // 2D filtering of color images is a little different
1425            // than
1426                // of mono images
1427            int kcol;
1428            int leftBound = -halfK * 4;
1429            int rightBound = halfK * 4;
1430            if (is2D) {
1431                for (row = -halfK; row <= halfK; row++) { // go through all rows
1432                    for (col = leftBound, kcol = -halfK; col <= rightBound; col += 4, kcol++) {
1433                        // go through every 4th column
1434                        if (kernel[kCenter+kcol+row*kernelSize] != 0) { // but don't bother
1435                            // copying
1436                            // want
1437                            // into the list if we don't
1438                            // that element (the kernel's
1439                            // pixel is zero)
1440                            kernelMask[count++] = data[i+col+row*sliceWidth];
1441                        }
1442                    }
1443                }
1444            }
1445        }

```

```

1432             }
1433         }
1434     }
1435     else {           // find neighbors in a volume
1436         int slice;
1437         // halfK-number of kernelSize slices (to get to the center slice)
1438         for (slice = -halfK; slice <= halfK; slice++) {
1439             for (row = -halfK; row <= halfK; row++) {
1440                 for (col = leftBound, kcol = -halfK; col <= rightBound; col += 4, kcol++) {
1441                     if (kernel[kCenter+kcol+row*kernelSize+slice*kernelSize*kernelSize]
1442                         != 0) {
1443                         kernelMask[count++] =
1444                             data[i+col+row*sliceWidth+slice*sliceWidth*height];
1445                     }
1446                 }
1447             }
1448         }
1449     }
1450 }
1451 else { // a mono image
1452     if (is2D) {
1453         for (row = -halfK; row <= halfK; row++) { // go through all rows
1454             for (col = -halfK; col <= halfK; col++) { // go through all columns
1455                 if (kernel[kCenter+col+row*kernelSize] != 0) { // but don't bother
1456                     copying
1457                         // into the list if we don't
1458                         // want that element (the
1459                         // kernel's pixel is zero)
1460                     kernelMask[count++] = data[i+col+row*width];
1461                 }
1462             }
1463         }
1464     }
1465     else {           // find neighbors in a volume
1466         int slice;
1467         // halfK-number of kernelSize slices (to get to the center slice)
1468         for (slice = -halfK; slice <= halfK; slice++) {
1469             for (row = -halfK; row <= halfK; row++) {
1470                 for (col = -halfK; col <= halfK; col++) {
1471                     if (kernel[kCenter+col+row*kernelSize+slice*kernelSize*kernelSize] !=
1472                         0) {
1473                         kernelMask[count++] = data[i+col+row*width+slice*width*height];
1474                     }
1475                 }
1476             }
1477         }
1478     }
1479     return (kernelMask);
1480 }
1481
1482 /**
1483 * Sorts a list of values. Taken from Numerical Recipes in C, 2nd ed. William H.
1484 Press, et
1485 * al, page 332. Chose shell sort over a quicksort because both shell and quick
1486 are about the
1487 * same speed for the middle range of sizes of the list. The list is more likely
1488 during a

```

---

```

1487     * slice-filter operation to be smaller than the maximum 121 length. The list
1488     could be as
1489     * much as 1331 elements, but according to Numerical Recipes, it still runs fast
enough at
1490     * only N**1.25 an average for N < 60000). My guess is that sliceFilter is more
useful than
1491     * a volumeFilter and will be plenty fast enough to not necessitate a quicksort for
a
1492     *
1493     * @param float a[]      The list to sort.
1494     */
1495     private final void shellSort(float a[]) {
1496         int N = a.length - 1;
1497         int i, j;
1498         int inc = 1;
1499         float val;
1500
1501         do {
1502             inc *=3;
1503             inc++;
1504         } while (inc <= N);
1505         do {
1506             inc /=3;
1507             for (i = inc + 1; i <= N; i++) {
1508                 val = a[i];
1509                 j = i;
1510                 while (a[j - inc] > val) {
1511                     a[j] = a[j - inc];
1512                     j -= inc;
1513                     if (j <= inc) break;
1514                 }
1515                 a[j] = val;
1516             }
1517         }while (inc > 1);
1518     }
1519
1520 /**
1521     * Finds the median value of the list. Median assumes the list of values starts
at index 1,
1522     * not an index of 0.
1523     * (i.e., 1st element is not included.)
1524     * @param listList of numbers
1525     * @return The median.
1526     * @author parsonsd
1527     */
1528     private final float median(float list[]) {
1529         int N;
1530         float med;
1531
1532         N = list.length - 1;
1533
1534         if ((N%2) != 0) {
1535             med = list[N/2];
1536         }
1537         else {
1538             med = (list[N/2] + list[N/2+1])/2;
1539         }
1540         return (med);
1541     }

```

```

1542
1543
1544     /**
1545      * Finds the mean value (average) in the list. Mean assumes the list of values
1546      starts at
1547      * index 1, not an index of 0.
1548      * (i.e., 1st element is not included.)
1549      * @param listList of numbers
1550      * @return floatThe mean.
1551      * @author parsonsd
1552      */
1553     private final float mean(float list[]) {
1554         int i;
1555         float sum = 0;
1556
1557         for (i = 1; i < list.length; i++) {
1558             sum += list[i];
1559         }
1560         return (float)(sum/(list.length - 1));    // length-1 because list goes from [1 ...
1561     }
1562
1563     /**
1564      * Finds the standard deviation of the values in the input list
1565      * (defined as: s = [(1/(N-2))*SUM (from 1 to N-1)[ (Xi - <b>x</b>)
1566      * <b>)^2]]^(1/2))
1567      * @param listThe list of numbers.
1568      * @param average Arithmetic mean of the values in list.
1569      * @return The standard deviation.
1570      */
1571     private final float standardDeviation(float list[], float average) {
1572         int i;
1573         int N = list.length;
1574
1575         double sum = 0.0;
1576
1577         for (i = 1; i < N; i++) {
1578             sum += (list[i] - average)*(list[i] - average);
1579         }
1580         return ((float) Math.sqrt(sum/(N-2)));        // sqrt((1/(N-2)) * sum)
1581     }
1582
1583     /**
1584      * Creates the standard progressBar. Stores in the class-global, progressBar.
1585      */
1586     private void buildProgressBar(){
1587         try {
1588             if (pBarVisible == true) {
1589                 progressBar = new ViewJProgressBar(srcImage.getImageName(), "Filtering image ...",
1590                     0, 100, true, this, this);
1591                 int xScreen = Toolkit.getDefaultToolkit().getScreenSize().width;
1592                 int yScreen = Toolkit.getDefaultToolkit().getScreenSize().height;
1593                 progressBar.setLocation(xScreen/2, yScreen/2);
1594                 progressBar.setVisible(true);
1595             }
1596         } catch (NullPointerException npe) {
1597             if (Preferences.isDebugEnabled()) {
1598                 Preferences.debug("AlgorithmMedian: NullPointerException found while building
1599                             progress bar.");
1600             }
1601         }
1602     }

```

```
1599         }
1600     }
1601 }
1602 /**
1603  * If the progress bar is visible, sets the text to: <br><tt>Copying all <i>color</i>
1604  * values
1605  * ... </tt>
1606  * @param colorTextThe color to use. E.g., "red" or "blue".
1607 */
1608 private void setCopyColorText(String colorText)
1609 {
1610     try {
1611         if (pBarVisible == true) {
1612             progressBar.setMessage("Copying all " + colorText + " values ... ");
1613         }
1614     } catch (NullPointerException npe) {
1615         if (Preferences.isDebugEnabled()) {
1616             Preferences.debug("AlgrithmMedian: NullPointerException found while setting
progress bar text.");
1617         }
1618     }
1619 }
```